

## PATENT ABSTRACTS OF JAPAN

(11)Publication number : 10-232779

(43)Date of publication of application : 02.09.1998

(51)Int.Cl. G06F 9/38  
G06F 9/30  
G06F 9/45

(21)Application number : 10-048480

(71)Applicant : TEXAS INSTR INC &lt;TD&gt;

(22)Date of filing : 23.01.1998

(72)Inventor : SIMAR LAURENCE R  
SESHAN NATARAJAN  
TATGE REID E  
DAVIS ALAN L

(30)Priority

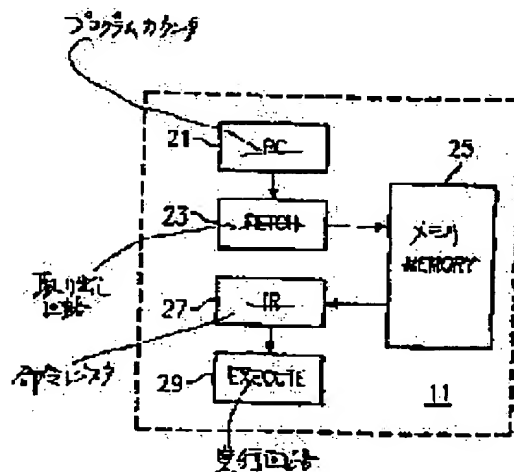
Priority number : 97 36223 Priority date : 24.01.1997 Priority country : US

## (54) METHOD AND DEVICE FOR PARALLEL INSTRUCTION PROCESSING

(57)Abstract:

PROBLEM TO BE SOLVED: To provide a data processing system with which one part of instructions is checked and it can be designated whether the instructions are to be executed parallelly or successively.

SOLUTION: A data processing circuit 11 has a circuit (program counter 21, program memory 23 and memory 25) for generating one set of instructions and one part of each instruction includes the designation showing whether the other instruction in one set of instructions and that instruction can be simultaneously executed or not. A program execution circuit 29 receives one set of instructions, selectively responds to one part of instructions and simultaneously executes the plural instructions shown by that one part of instructions.



## LEGAL STATUS

[Date of request for examination]

[Date of sending the examiner's decision of rejection]

[Kind of final disposal of application other than the examiner's decision of rejection or application converted registration]

[Date of final disposal for application]

[Patent number]

[Date of registration]

[Number of appeal against examiner's decision  
of rejection]

[Date of requesting appeal against examiner's  
decision of rejection]

[Date of extinction of right]

Copyright (C); 1998,2003 Japan Patent Office

(19) 日本国特許庁 (J P)

(12) 公開特許公報 (A)

(11) 特許出願公開番号

特開平10-232779

(43) 公開日 平成10年(1998) 9月2日

(51) Int. Cl. <sup>6</sup>	識別記号	F I
G 0 6 F 9/38	3 7 0	G 0 6 F 9/38 3 7 0 X
9/30	3 5 0	9/30 3 5 0 F
9/45		9/44 3 2 2 F

審査請求 未請求 請求項の数3 O L 外国語出願 (全 48 頁)

(21) 出願番号 特願平10-48480

(22) 出願日 平成10年(1998) 1月23日

(31) 優先権主張番号 0 3 6 2 2 3

(32) 優先日 1997年1月24日

(33) 優先権主張国 米国 (U S)

(71) 出願人 590000879

テキサス インストルメンツ インコーポ  
レイテッド

アメリカ合衆国テキサス州ダラス、ノース  
セントラルエクスプレスウェイ 13500

(72) 発明者 ローレンス アール. シマー

アメリカ合衆国 テキサス州リッチモン  
ド, モートン リーグ ロード 2103

(72) 発明者 ナタラジャン セスハン

アメリカ合衆国 テキサス州ヒュースト  
ン, エラ リー レーン 9550 ナンバー  
408

(74) 代理人 弁理士 浅村 皓 (外 3 名)

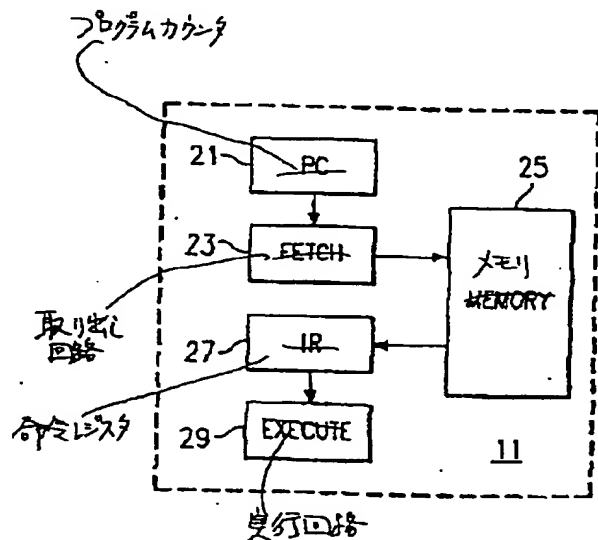
最終頁に続く

(54) 【発明の名称】 命令並列処理方法及び装置

(57) 【要約】

【課題】 命令の一部を検査して、命令を並列実行するか逐次実行するかを指定できるデータ処理システムを開示する。

【解決手段】 データ処理回路11には、1組の命令を発生させる回路(プログラムカウンタ21, プログラムメモリ23, メモリ25)があり、各命令の1部分に、その1組の命令の中の別の命令と、その命令とを同時に実行できるか否かを示す指定が含まれている。プログラム実行回路29は、その1組の命令を受信し、命令の1部分に選択的に応答して、その命令の1部分によって示される複数の命令を同時に実行する。



**【特許請求の範囲】**

【請求項 1】 それぞれの命令を同時に実行することができるか否かを示す、それぞれの命令の 1 部分を含む 1 組の命令を発生する回路を有するデータ処理回路を含むデータ処理システムであって、

前記データ処理回路は、前記発生回路に接続され、前記 1 組の命令を受信するプログラム実行回路を含みかつ前記命令の 1 部分に選択的に応答して前記命令の 1 部分によって示される複数の前記命令を同時に実行するデータ処理システム。

【請求項 2】 データ処理システムにおいてプログラム命令を処理する方法であって、

その命令の中の別の命令と同時にそれぞれの命令を実行できるか否かを示すそれぞれの命令の 1 部分を、1 組の命令に供給するステップと、

複数の前記命令を同時に実行できるか否かを、前記命令の 1 部分から決定するステップと、を含む方法。

【請求項 3】 データ処理システムによって実行するため、プログラムをコンパイルする方法であって、プログラムの中の第 1 のプログラム命令の直後に続く第 2 のプログラム命令と同時に、その第 1 のプログラム命令を実行できるか否かを決定するステップと、

第 1 の命令を第 2 の命令と同時に実行できるか否かを示す命令の 1 部分を、その第 1 の命令に供給するステップと、を具備する方法。

**【発明の詳細な説明】****【0001】**

【発明の属する技術分野】 本発明は、一般にデータ処理に関し、より詳細には、プログラム命令の並列実行および逐次実行の両方を備えたデータ処理に関する。

**【0002】**

【発明が解決しようとする課題】 データ処理システムおよびデータプロセッサは、實際上生活のあらゆる側面に影響を与える多数の用途で使用されている。関連する処理システムおよびデータプロセッサの速度およびスループットを速くすることにより、これらの多数の用途の有用性を向上させることができるのが普通である。

【0003】 速度およびスループットを向上させることが可能な場合の 1 つの方法は、プログラム命令を逐次的に実行するよりむしろ並列に実行することである。この点に関する公知の方法の 1 つは、プログラム命令の並列実行または逐次実行を指定する特殊なモードの命令を使用することである。この点に関する公知の別の方法は、並列型パケットにヌル命令を指定するためにマスクを使用することである。この点に関する公知の別の解決方法は、並列モードだけでデータ処理を実行することである。

【0004】 前述の手法は速度およびスループットを改善することができるが、実現することがかなり難しいために望ましくない上、処理に伴うオーバーヘッドの点で

費用がかかるという欠点がある。

【0005】 したがって、上に説明した方法に付随する実現上の困難と、処理に伴うオーバーヘッドとが減少するように、プログラム命令を並列で実行できるようにすることが望ましい。

【0006】 本発明の目的は、与えられたプログラム命令の一部を利用して、その命令を他のプログラム命令と同時に実行することができるか否かを決定することである。

**【0007】**

【課題を解決するための手段】 一般に、そしてまた本発明の形式においては、1 組の命令を発生する回路をもつデータ処理装置が用意されており、これらの命令には、それぞれ該当する命令を、その複数の命令の中の別の命令と同時に実行できるか否かを示す命令の 1 部分が含まれている。このデータ処理回路には、前記発生回路に接続されていて、その 1 組の命令を受信すると、インジェクタ部に応答して複数の命令を同時に実行するプログラム実行回路が含まれている。

【0008】 本発明の別の形式では、データ処理装置の中の中央処理装置 (CPU) を動作させる方法は、それぞれ該当する命令を同時に実行できるか否かを示すため、1 組の命令にそれぞれの命令の 1 部分を与えるステップと、その命令の 1 部分から複数の命令を同時に実行できるか否かを決定するステップとから構成されている。

【0009】 本発明の別の実施例は、説明および図面から明らかになるであろう。

**【0010】**

【発明の実施の形態】 添付の図面において、異なる図面および表における対応する番号と符号は、とくに断りのない限り対応する部品を示している。

【0011】 図 1 は、本発明によるデータ処理システム 10 のブロック図である。データ処理システム 10 にはデータ処理回路 11 および周辺回路 13, 15, 17, 19 が含まれている。図 1 の代表的実施例において、データ処理回路 11 は、データ処理回路 1.1 と周辺回路 13, 15, 17, 19 との間で情報を転送するため、周辺回路 13, 15, 17, 19 のそれぞれに接続されている。しかし、以下の説明から明らかになるとおり、本発明によるデータ処理システムは、この説明の前で後でも当業者ならば想像できるように、相互に接続されかつデータ処理回路 11 に接続されるいかなる量の、また（たとえば周辺装置 13, 15, 17, 19 など）いかなる形式の周辺回路および周辺装置が含まれていてもよい。

【0012】 図 2 は、図 1 のデータ処理回路 11 の 1 つの代表的実施例の一部を示している。図 2 において、取り出し回路 23 が、プログラムカウンタ 21 によって指定されるアドレスでメモリ 25 にアクセスすると、命令

パケットがそのアドレスで命令レジスタ 27 にロードされる。プログラム実行回路 29 は、命令レジスタ 27 に保持されているその命令パケットの中の複数の命令を復号化して実行する。

【0013】図 3 は、メモリ 25 から取り出された命令パケットの基本フォーマットを示している。開示した代表的実施例においては、命令パケットには 4 個の 32 ビット命令 A, B, C, D が含まれている。図 3 に示すように、命令 A, B, C, D はメモリ 25 の連続したアドレスに格納されている。したがって、正常のプログラム命令の逐次実行中、命令 A が最初に実行され、命令 B, C, D が順番に続いて実行される。

【0014】図 3 の各命令のビット 0 は p- ビットと呼ばれている。p- ビットは、命令の実行方法を定義する。図 3 の命令パケットの p- ビットは、プログラム実行回路 29 によって左から右に検査される。ある命令の p- ビットが論理 1 に等しい場合、そのパケットの中の次の順番の命令が、最初に言及した命令と同時に実行される。プログラム実行回路 29 は、その命令パケットの中の p- ビットが論理 0 に等しい命令に到達するまでこの規則を適用する。

【0015】ある命令の p- ビットが 0 の場合、その命令の後に（そしてその命令と並列に実行されるすべての命令の後に）続いて次の順番の命令が実行される。プログラム実行回路 29 は、命令パケットの中の p- ビットが論理 1 の命令に到達するまでこの規則を適用する。

【0016】図 4 から図 19 は、上に説明した p- ビットの規則の応用例を示している。

【0017】図 4 は、すべての p- ビットが 0 の命令パケットを示している。したがって、命令 A から命令 D は図 5 に示すとおり、順番に連続して実行される。

【0018】図 6 は、命令 A, B, C の p- ビットが 1 に等しく、命令 D の p- ビットが 0 の命令パケットを示している。したがって、命令 A, B, C, D は同時に実行される。すなわち、図 7 に示すとおり並列で同時に実行される。

【0019】図 8 の命令においては、命令 C の p- ビットだけが 1 にセットされているので、図 9 の実行順序になる。すなわち、命令 A, B が逐次実行され、並列に実行される命令 C, D が、その後に続いて実行される。

【0020】図システム 10 の命令パケットにおいては、命令 B の p- ビットだけが 1 にセットされているので、図 11 の実行順序になる。すなわち、命令 A が実行され、それに続いて命令 B, C が並列に実行され、その後に続いて命令 D が実行される。

【0021】図 12 の命令パケットにおいては、命令 B, C の p- ビットが 1 にセットされ、命令 A, D の p- ビットはゼロである。このようになっていると、図 13 に示す命令順になる。すなわち、命令 A が実行され、それに続いて命令 B, C, D が並列に実行される。

【0022】図システム 14 の命令パケットにおいては、命令 A の p- ビットだけが論理 1 にセットされているので、図 15 の実行順序になる。すなわち、命令 A, B が並列で実行され、その後に続いて命令 C が実行され、最後に命令 D が実行される。

【0023】図 16 の命令パケットにおいては、命令 A, C の p- ビットが 1 にセットされ、命令 B, D の p- ビットはゼロになっているので、図 17 に示す実行順序になる。すなわち、命令 A, B が並列に実行され、それに続いて命令 C, D が並列に実行される。

【0024】図 18 の命令パケットにおいては、命令 A, B の p- ビットが 1 にセットされ、命令 C, D の p- ビットはゼロである。このようになっていると、図 19 に示す実行順序になる。すなわち、命令 A, B, C が並列に実行され、その後に続いて命令 D が実行される。

【0025】開示した例の中の命令パケットには、4 個のプログラム命令が含まれているので、プログラムコンパイラは常に命令 D（4 番目の命令）に、0 の p- ビットを与えてもよい。コンパイラは、命令 A, B を並列で実行する規則（propriety）、命令 B, C を並列で実行する規則、命令 A, B, C を並列で実行する規則に基づいて、命令 A, B, C の残りの p- ビットの値を決定する。たとえば、命令 B の実行に、命令 A の実行結果が必要な場合、コンパイラは、命令 B が命令 A の後に実行されるように、命令 A に 0 の p- ビットを与えるであろう。別の例として、命令 B, C が同一レジスタをアクセスする場合、コンパイラは、命令 B, C が並列ではなく、逐次的に実行されることを保証するため、命令 B に 0 の p- ビットを与えるであろう。

【0026】図 20 は、本発明の一実施例を使用するマイクロプロセッサ 1 のブロック図である。マイクロプロセッサ 1 は、VLIW デジタル信号プロセッサ（“DSP”）である。明瞭にするため、図 20 は、マイクロプロセッサ 1 の、本発明の実施例を理解するために関連する部分だけを示している。DSP の一般的な構造の詳細は公知であり、どこかほかでも容易に調べ出すことができるであろう。たとえば、Frederick Boutaud ほかに発行された米国特許第 5,072,418 号は、DSP を詳細に説明している。ここでこの特許に言及することにより、この特許の開示内容を本願に組み入れることにする。Gary Swoboda ほかに発行された米国特許第 5,329,471 号は、DSP のテストとエミュレーションの方法を詳細に説明している。ここでこの特許に言及することにより、この特許の開示内容を本願に組み入れることにする。マイクロプロセッサ技術に関する当業者が本発明を評価して使用できるようにするため、マイクロプロセッサ 1 の、本発明に関連する部分を以下に十分詳細に説明する。

【0027】マイクロプロセッサ 1 には、中央処理装置（CPU）10、データメモリ 22、プログラムメモリ

23、周辺回路60およびダイレクトメモリアクセス(DMA)を備えた外部メモリアンタフェース(EMIF)61がある。さらにCPU10には、命令取り出し/復号化ユニット10a~10c、演算およびロード/ストアユニットD1、乗算器M1、ALU/シフターユニットS1、演算論理ユニット("ALU")L1、データを読み出しかつデータを書き込む共用マルチポートレジスタファイル20aを含む複数の実行ユニットがある。復号化された命令は、示されていないいろいろな制御ラインの組を介して、命令取り出し/復号化ユニット10a~10cから、機能ユニットD1、M1、S1、L1に与えられる。データは、第1組のバス32aを介してロード/ストアユニットD1と、第2組のバス34aを介して乗算器M1と、第3組のバス36aを介してALU/シフターユニットS1と、第4組のバス38aを介してALUL1と、レジスタファイル20aとの間で送受される。データは、第5組のバス40aを介して、メモリ22とロード/ストアユニットD1との間で送受される。上に説明した全データ経路は、レジスタファイル20bおよび実行ユニットD2、M2、S2、L2と2重化されていることに注意されたい。命令は、1組のバス41を介して、取り出しユニット10aによって、命令メモリ23から取り出される。外部のテストシステム51によって制御されるエミュレーションユニット50は、集積回路の内部動作にアクセスできるようにする。

【0028】メモリ22、23は、マイクロプロセッサ1の集積回路の一部として図20に示されており、その範囲はボックス42で示されていることに注意されたい。メモリ22、23は、マイクロプロセッサ1の集積回路42の外部にあってもよいし、あるいはメモリの一部が集積回路42の内部にあり、そしてメモリの一部が集積回路42の外部にあってもよい。また別の数の実行ユニットを使用してもよい。

【0029】マイクロプロセッサ1がデータ処理システムに組み込まれると、図1に示すように、メモリまたは周辺装置を追加してマイクロプロセッサ1に接続することができる。たとえば、外部バス73を介して、ランダムアクセスメモリ(RAM)70、読み出し専用メモリ(ROM)71およびディスク72が接続されていることが示されている。バス73は、マイクロプロセッサ42の内部の機能ブロック61の重要な部分である外部メモリアンタフェース(EMIF)に接続されている。ダイレクトメモリアス(DMA)コントローラもブロック61に含まれている。DMAコントローラは、一般にマイクロプロセッサ1の内部のメモリと周辺回路との間でデータを転送するために使用されるとともにメモリと、マイクロプロセッサ1の外部にある周辺装置との間でデータを転送するために使用される。

【0030】本発明の側面から利点を得ることができる

いくつかの代表的なシステムは、その開示内容を本願に組み入れた米国特許第5,072,418号に、特にこの特許の図2から図18を参照して説明されている。米国特許第5,072,418号に説明されているシステムをさらに改善するため、性能を改善しかつ原価を低減するため本発明の側面を組み入れたマイクロプロセッサを使用してもよい。そのようなシステムには、工業におけるプロセス制御、自動車(automotive vehicle)システム、モータ制御、ロボット制御システム、衛星通信システム、反響消去システム、モデム、ビデオ映像システム、音声認識システム、暗号付きボコーダーモデムシステムなどが含まれるが、それらに限定されるものではない。

【0031】図20のマイクロプロセッサの各種アーキテクチャの特徴は、共同譲渡された特許出願シリアル番号60/036,482(TI控え番号T-25311)に説明されている。図20のマイクロプロセッサの命令の完全な命令セットも、共同譲渡された特許出願シリアル番号60/086,482(TI控え番号T-25311)に説明されている。

【0032】図21は、図20のマイクロプロセッサの実行ユニットおよびレジスタファイルのブロック図であり、各種機能ユニットを接続するバスの詳細な図を示している。本図において、特に注意しない限り、全データバスは32ビット幅である。バス40aには、マルチプレクサ200aによって駆動されるアドレスバスDA1がある。このため、ロード/ストアユニットD1またはD2によって発生したアドレスを、レジスタファイル20aに対するロードまたはストアのアドレスにすることができる。データバスLD1は、アドレスバスDA1によって指定されたメモリのアドレスからのデータを、ロードユニットD1のレジスタにロードする。ユニットD1は、与えられたデータをレジスタファイル20aにストアする前に操作(manipulate)する。同様にデータバスST1は、レジスタファイル20aからのデータをメモリにストアする。ロード/ストアユニットD1は次の演算、すなわち、32ビットの加算、32ビットの減算、線形および循環アドレス計算(linear and circular address calculations)を実行する。ロード/ストアユニットD2は、アドレスを選択するマルチプレクサ200bの支援により、D1と同様に動作する。

【0033】ALUユニットL1は次のタイプの演算、すなわち、32/40ビット数値演算および比較演算、32ビットの最左端の1ビットおよび0ビットの計数(leftmost 1, 0, bit counting for 32 bits)、32ビットおよび40ビットの正規化回数の計数および論理演算を実行する。ALUL1には、32ビットのソースオペランドに対する入力src1と、第2の32ビットのソースオペランドに対する入力src2とがある。入力msb\_srcは、40ビットのソースオペランドを生成するために使用される8ビットの値である。ALUL1には、32ビットの宛先オペランドに対する出力dstがある。出

カmsb\_dst は、40ビット宛先オペランドを形成するために使用される8ビットの値である。レジスタファイル20aの中にある2つの32ビットレジスタは、40ビットのオペランドを保持するために連結されている。マルチプレクサ211は入力src1に接続されており、32ビットのオペランドが、バス38aを介してレジスタファイル20aから、またはバス210を介してレジスタファイル20bから取得されることを可能にしている。マルチプレクサ212は入力src2に接続されており、32ビットオペランドが、バス38aを介してレジスタファイル20aから取得されること、またはバス210を介してレジスタファイル20bから取得されることを可能にしている。ALUユニットL2はユニットL1と同様に動作する。

【0034】ALU/シフターユニットS1は次のタイプの演算、すなわち、32ビット数値演算、32/40ビットシフトおよび32ビットビットフィールド演算、32ビット論理演算、分岐および定数の発生を実行する。ALU S1には、32ビットのソースオペランドに対する入力src1と、第2の32ビットのソースオペランドに対する入力src2とがある。入力msb\_src は、40ビットのソースオペランドを形成するために使用される8ビットの値である。ALU L1には、40ビットの宛先オペランドに対する出力dstがある。出力msb\_dst は、40ビットの宛先オペランドを生成するために使用される8ビットの値である。マルチプレクサ213は、入力src2に接続されており、32ビットのオペランドが、バス36aを介してレジスタファイル20aから取得されること、またはバス210を介してレジスタファイル20bから取得されることを可能にしている。ALUS2は、ユニットS1と同様に動作するが、これに加えて制御レジスタファイル102との間の両方向でレジスタ転送を実行する。

【0035】乗算器M1は16×16の乗算を実行する。ALU S1には、32ビット宛先オペランドに対する出力dstがある。乗算器M1には、32ビットのソースオペランドに対する入力src1と、32ビットの宛先オペランドに対する入力src2とがある。マルチプレクサ214は入力src2に接続されており、32ビットオペランドが、バス34aを介してレジスタファイル20aから取得されること、またはバス210を介してレジスタファイル20bから取得されることを可能にしている。乗算器M2は、乗算器M1と同様に動作する。

【0036】図22は、図1のマイクロプロセッサにおける命令実行パイプラインの処理フェーズを示す図表である。各フェーズは、システムクロックのクロックサイクルにほぼ対応している。たとえば、マイクロプロセッサ1が200メガヘルツで動作している場合、各フェーズは通常5ナノ秒である。しかし、RAM70のようなメモリあるいは周辺装置からデータが期待されている場

合、期待されているときにデータが用意されていないとパイプラインはストール(stall)する。パイプラインがストールすると、いくつかのシステムクロックサイクルの間、あるパイプラインフェーズになっている。

【0037】図22において、命令を処理する第1フェーズは、フェーズPGでプログラムアドレスを発生させることである。これは、制御レジスタファイル102に配置されているプログラム取り出しカウンタPFCをローディングすることにより実行される。第2の命令処理フェーズPS中に、バス41の役割の1つであるプログラムアドレスバスPADDRを介して、命令取り出しパケットのアドレスがプログラムメモリ23に送られる。第3のフェーズPWは、メモリ23におけるアクセス時間を考慮するための待ち合わせフェーズである。第4のフェーズPR中に、バス41の一部であるデータバスPDATA\_1を介してプログラムメモリ23からプログラム取り出しパケットを使用することができる。第5の処理フェーズDP中に命令の並列性が検出されると、実行できる命令が適切な機能ブロックにディスパッチされる。パイプライン動作のこの側面については、後の節でさらに詳細に説明する。第6の処理フェーズDC中に、実行可能な命令が復号化され、制御信号が発生して、各種データ経路と機能ユニットとを制御する。

【0038】図23は、図1のマイクロプロセッサ1における命令実行パイプラインの実行フェーズを示す図表である。第1の実行フェーズE1中に、“ISC”と呼ぶ1サイクル命令と、“BR”と呼ぶ分岐命令が終了する。指定された実行ユニットは、制御回路100によって指示されるとおり、図23に示す演算を実行する。第2の実行フェーズE2中に、制御回路100の制御によって指定された実行ユニットにより、次の命令、すなわち、整数の乗算(IMPY)、プログラムストア命令(STP)およびデータストア命令(STD)を終了させる。第3の実行フェーズE3中に、データメモリシステム(DMS)からのデータをラッチすることにより、指示されたとおり、ロードデータ命令(LD)の実行を継続する。第4の実行フェーズE4中に、実行ユニットD1または実行ユニットD2のデータ入力レジスタDDATA\_1に、E3でラッチされたデータが転送される。第5の実行フェーズE5中に、レジスタDDATA\_1のデータを操作して、その操作したデータを、レジスタファイル20aまたは20bの中の指定されたレジスタに書き込むことにより、LD命令が終了する。

【0039】図24は、図22の処理フェーズ中に命令取り出しパケットを処理するタイミングの詳細と、図23の実行フェーズ中の実行パケットの実行とを示すタイミング図である。フェーズPSではプログラムメモリレディ信号PRDYがローのため、フェーズPWにパイプストールが示され、フェーズE2ではデータメモリレディ信号DRDYがローのため、フェーズE3に第2のパ

イブストールが示されていることに注意されたい。

【0040】図25は、図20のマイクロプロセッサにおける命令のディスパッチ動作を示すブロック図である。本実施例において、命令取り出しパケットは8つの命令を含んでいる。命令取り出しパケット1710は、図示の通り8つの実行ユニットにディスパッチされて復号化される。取り出しパケット1720は分岐命令1725を含んでいる。取り出しパケット1730は3つの命令実行パケットを含んでいる。第1の実行パケットは、2つの命令、ZERO、SHLを含んでおり、これらの命令は、分岐命令1725に関する第1遅延スロットにおける処理を開始する。第2の実行パケットは、4つの命令、ADD、SIB、STW、STWを含んでおり、これらの命令は、分岐命令1725に関する第2遅延スロットにおける処理を開始する。第3の実行パケットは、2つの命令、ADDK、BRを含んでおり、これらの命令は、分岐命令1725の第3遅延スロットの処理を開始する。

#### 【0041】並列演算

命令は常に1度に8つ取り出される。これが1つの取り出しパケットを構成する。取り出しパケットの基本フォーマットは、図26に示されている。取り出しパケットの実行のグループ化は、各命令のビット0のp-ビットによって指定される。取り出しパケットは8語に揃えられている。

【0042】p-ビットは、命令の並列実行を制御する。p-ビットは左から右へ（低いアドレスから高いアドレスへ）走査される。命令iのp-ビットが1の場合、命令i+1は命令iと（命令iと同じサイクルで）並列に実行されることになる。命令iのp-ビットが0の場合、命令iの後のサイクルで命令i+1が実行される。並列で実行する全命令は、1つの実行パケットを構成する。1つの実行パケットは最大8つの命令を含むことができる。実行パケットの中の全命令は独自の機能ユニットを使用しなければならない。

【0043】実行パケットは、8語の境界を横切ることにはできない。したがって、取り出しパケットの最後のp-ビットは常に0にセットされており、各取り出しパケットは、新しい実行パケットを開始させる。図4から図19について考察したとおり、取り出しパケットに対し

```
ADD S1 A0, A1, A2 ; \
| | SHR .S1 A3, 15, A4 ; /
```

【0049】次の実行パケットは有効である。

```
ADD L1 A0, A1, A2 ; \
| | SHR .S1 A3, 15, A4 ; /
```

【0050】相互経路(Cross Paths) (1Xおよび2X)

実行ユニットごとの(S, L, Mのいずれか)1つのユニットは、データ経路ごとに相互経路(1Xおよび2X)を介してその反対側のレジスタファイルからソース

で3タイプのp-ビットのパターンがある。これらの3つのp-ビットパターンは、次に示す8つの命令の実行順序になる。すなわち、完全シリアル、完全並列、部分的シリアルの3つである。

#### 【0044】並列符号の例

文字“||”は、命令が直前の命令と並列に実行されることを表している。図27の取り出しパケットにおいて、この符号は次のように表示される。

【表1】

	命令A
	命令B
	命令C
	命令D
	命令E
	命令F
	命令G
	命令H

【0045】実行パケットの中間に対する分岐動作  
実行パケットの中間に対する分岐が発生すると、それよりも低いアドレスの全命令は無視される。図27の例において、命令Dを含むアドレスに対する分岐が発生すると、DおよびEだけが実行される。命令Cも同一実行パケットの中にあるが、命令Cは無視される。また命令A、Bは時間的に前の方の実行パケットにあるから、命令A、Bも無視される。

#### 【0046】資源の制約

同一実行パケットの中のどの2つの命令も同一資源を使用することはできない。また、どの2つの命令も、同一サイクル中に同一レジスタに書き込むことはできない。以下の節において、命令が使用できる資源のそれぞれについて説明する。

#### 【0047】機能ユニット

同一機能ユニットを使用する2つの命令を同一実行パケットで発行することはできない。

【0048】次の実行パケットは無効である。

【表2】

.S1が両命令に使用されている。

【表3】

2つの異なる機能ユニットが使用されている。

オペランドを読み出すことができる。たとえば、S1は、Aレジスタファイルから両方のオペランドを読み出すことができるか、あるいは1X相互経路を使用して、Bレジスタファイルから1つのオペランドを読み出すことができる。このことは、ユニット名の後にくるXによ



って表示される。

【0051】AからBの間およびBからAの間には、それぞれ1つの経路しかないのであるから、レジスタファイル間で同一X相互経路を使用する2つの命令を同一実行

```
ADD. L1X  A0, B1, A1  ; \
|| MPY. M1X  A4, B4, A5  ; /
```

【0053】次の実行パケットは有効である。

```
ADD. L1X  A0, B1, A1  ; \
|| MPY. M2X  A4, B4, 132 ; /
```

【0054】命令フィールドのxビットがセットされている場合、オペランドは宛先と反対のレジスタファイルから来る。

【0055】ロード経路およびストア経路

ロードおよびストアは、他のレジスタファイルへのロードまたは他のレジスタファイルからのストアを実行しながら、1つのレジスタファイルからのアドレスポインタ

```
LDW D1  *A0, A1  ; \
|| LDW. D1  *A2, B2  ; /
```

次の実行パケットは有効である。

```
LDW D1  *A0, A1  ; \
|| LDW. D2  *B0, B2  ; /
```

【0057】同一レジスタファイルにロード中の、および/あるいは同一レジスタファイルからストア中の2つのロードおよびストアを、同一実行パケットで発行する

```
LDW D1  *A4, A5  ; \
|| STW. D2  A6, *B4  ; /
```

【0059】次の実行パケットは有効である。

```
LDW D1  *A4, B5  ; \
|| STW. D2  A6, *B4  ; /
```

【0060】長い経路(Long Path)

1サイクル当たり、1つだけの長い結果(long result)を両側のレジスタファイルに書き込むことができる。Sユニットおよび、Lユニットは、長いソースオペランドの読み出しレジスタポートと、長い結果の書き込みレジ

```
ADD. L1  A5:A4, A1, A3:A2  ; \
|| SHL. S1  A8, A9, A7:A6  ; /
```

次の実行パケットは有効である。

```
ADD. L1  A5:A4, A1, A3:A2  ; \
|| SHL. S2  B8, B9, B7:B6  ; /
```

【0062】、Lユニットおよび、Sユニットは、ストアポートと、これら両ユニット用の長い読み出しレジスタを共用しているので、ストアと同じ実行パケットで、長い値を読み出す演算を、Lユニットおよび/あるいは

```
ADD. L1  A5:A4, A1, A3:A2  ; \
|| STW. D1  A8, *A9  ; /
```

次の実行パケットは有効である。

```
ADD. L1  A4, A1, A3:A2  ; \
|| STW. D1  A8, *A9  ; /
```

【0064】レジスタ読み出し

同一サイクルで、同一レジスタに対する4回以上の読み

行パケットで発行することはできない。

【0052】次の実行パケットは無効である。

【表4】

両命令に対して1X相互経路が使用されている。

【表5】

これらの命令は、1Xおよび2X相互経路を使用している。

を使用することができる。同一レジスタファイルからのアドレスポインタを使用する2つのロードおよび/あるいはストアを同一実行パケットで発行することはできない。

【0056】次の実行パケットは無効である。

【表6】

同一レジスタファイルからのアドレスレジスタである。

【表7】

異なるレジスタファイルからのアドレスレジスタである。

ことはできない。

【0058】次の実行パケットは無効である。

【表8】

同一レジスタファイルとのロードおよびストアである。

【表9】

異なるレジスタファイルとのロードおよびストアである。

ストアポートとを共用しているので、実行パケットで、1側に1つだけこの種命令を発行してもよい。

【0061】次の実行パケットは無効である。

【表10】

; \ Aレジスタファイルへの  
; / 長い書き込みが2つある。

【表11】

; \ 各レジスタファイルへの  
; / 長い書き込みが1つある。

は、Sユニットに発行することはできない。

【0063】次の実行パケットは無効である。

【表12】

; \ 長い読み出し演算  
; / とストアがある。

【表13】

; \ ストアと一緒に長い  
; / 読み出しがない。

出しを発生させることはできない。条件付きレジスタはこの回数に含まれていない。

【0065】次のコードシーケンスは無効である。

```

      MPY      .M1  A1, A1, A4      ; レジスタ A1 を 5 回
| |  ADD      .L1  A1, A1, A5      読出している。
| |  SUB      .D1  A1, A2, A3

```

【0066】一方このコードシーケンスは有効である。

```

      MPY      .M1  A1, A1, A4      ; レジスタ A1 を 4 回
| |  [A1]  ADD      .L1  A0, A1, A5      だけ読出している。
      SUB      .D1  A1, A2, A3

```

【0067】レジスタ書き込み

同一レジスタに書き込む命令であって、異なる待ち合わせ時間をもつ命令が、異なるサイクルに発行される場合、同一サイクルで、同一レジスタに複数回の書き込みが発生することがある。たとえば、サイクル  $i+1$  の ADD の前のサイクル  $i$  で発行された MPY は、同一レジスタに書き込みをすることができない。その理由は、両命令がサイクル  $i+1$  で結果を書き込むからである。したがって、次のコードシーケンスは無効である。

【表 16】

```

L1:      ADD. L2      B5, B6, B7      ; 検出可能、競合する
| |      SUB. S2      B8, B9, B7
L2:      MPY. M2      B0, B1, B2      ; \ 検出不可可能
L3:      ADD. L2      B3, B4, B2      ; /
L4:  [!B1]  ADD. L2      B5, B6, B7      ; 検出可能、競合しない
| |  [B0]   SUB. S2      B8, B9, B7
L5:  [!B1]  ADD. L2      B5, B6, B7      ; \ 検出不可可能
| |  [B0]   SUB. S2      B8, B9, B7      ; /

```

【0069】パケット L2 の MPY とパケット L3 の ADD は、共に同時に B2 に書き込むが、分岐命令のため、L2 の後の実行パケットが、L3 以外の何らかのパケットになると、これは競合にならないであろう。したがって、L2 および L3 における潜在的な競合は、アセンブラによって検出されないかもしれない。L4 中の命令は書き込み競合を構成しない。何故かという、これらの命令は相互に排他的だからである。対照的に、L5 中の命令は相互に排他的であることが明確でないから、アセンブラは競合を決定することができない。同一レジスタに対して書き込みを複数回実行するコマンドをパイプラインが受信すると、その結果は定義されていない。

【0070】本発明の模範的な例が上に説明されているが、この実施例は本発明を限定するものではない。本発明は、多様な実施例の中で実現される。

【0071】関連特許に対する相互参照

本願は、本願と同時に出版され、共通譲渡された特許出願シリアル番号第 60/036,482 号 (T1 控え番号 T1-2135) に関連している。ここでこの特許出願に言及することにより、この特許出願の開示内容を本願に組み入れることにする。

【図面の簡単な説明】

【図 1】本発明によるデータ処理システムのブロック

【表 14】

レジスタ A1 を 5 回  
読出している。

【表 15】

レジスタ A1 を 4 回  
だけ読出している。

```

      MPY      .M1  A0, A1, A2
      ADD      .L1  A4, A5, A2

```

【0068】書き込み競合 (conflict) の検出可能性  
次の実行パケットのシーケンスは、異なる複数の書き込みの競合を示している。たとえば、実行パケット L1 中の ADD および SUB は、同一レジスタに書き込む。この競合は容易に検出できる。

【表 17】

図。

【図 2】図 1 のデータ処理回路の一部のブロック図。

【図 3】本発明で使用されている命令パケットの基本フォーマットを示す図。

【図 4】図 3 のフォーマットによる命令パケットの一例を示す図。

【図 5】図 4 の命令パケットによって定義される実行順序を示す図。

【図 6】図 3 のフォーマットによる命令パケットの別の例を示す図。

【図 7】図 6 の命令パケットによって定義される実行順序を示す図。

【図 8】図 3 のフォーマットによる命令パケットの別の例を示す図。

【図 9】図 8 の命令パケットによって定義される実行順序を示す図。

【図 10】図 3 のフォーマットによる命令パケットの別の例を示す図。

【図 11】図 10 の命令パケットによって定義される実行順序を示す図。

【図 12】図 3 のフォーマットによる命令パケットの別の例を示す図。

【図 13】図 12 の命令パケットによって定義される実行順序を示す図。

【図 14】図 3 のフォーマットによる命令パケットの別の例を示す図。

【図 15】図 14 の命令パケットによって定義される実行順序を示す図。

【図 16】図 3 のフォーマットによる命令パケットの別の例を示す図。

【図 17】図 16 の命令パケットによって定義される実行順序を示す図。

【図 18】図 3 のフォーマットによる命令パケットの別の例を示す図。

【図 19】図 18 の命令パケットによって定義される実行順序を示す図。

【図 20】本発明の実施例を使用するマイクロプロセッサのブロック図。

【図 21】図 20 のマイクロプロセッサの実行ユニットおよびレジスタファイルのブロック図。

【図 22】図 20 のマイクロプロセッサにおける命令実行パイプラインの処理フェーズを示す図表。

【図 23】図 20 のマイクロプロセッサにおける命令実行パイプラインの処理フェーズを示す図表。

【図 24】図 22 の処理フェーズ中に命令取り出しパケットを処理するタイミングと、図 23 の実行フェーズ中の実行パケットの実行とを示すタイミング図。

【図 25】図 20 のマイクロプロセッサにおける命令のディスパッチを示すブロック図。

【図 26】図 20 のマイクロプロセッサの命令取り出しパケットの基本フォーマットを示す図。

【図 27】部分的に並列な 8 語取り出しパケットを示す図。

【符号の説明】

1 マイクロプロセッサ

10 データ処理システム／中央処理装置 (CPU)

10a 命令取り出し

10b 命令ディスパッチユニット

10c 復号化ユニット

11 データ処理回路

13, 15, 17, 19 周辺回路

12a, 12b 演算およびロード／ストアユニット

14a, 14b 乗算器

16a, 16b ALU／シフターユニット

18a, 18b 数値・論理演算ユニット

20a, 20b レジスタファイル

21 プログラムカウンタ

22 データメモリ

23, 1723 プログラムメモリ

23 取り出し回路

25 メモリ

27 命令レジスタ

29 プログラム実行回路

32a, 34, 36a, 38a, 40a バス

41 バス

42 マイクロプロセッサの範囲を示すボックス

50 エミュレーションユニット

51 外部テストシステム

60 周辺回路

61 ダイレクトメモリアクセス (DMA) 付き外部メモリインタフェース

73 外部バス

70 RAM

71 ROM

72 ディスク装置

100 制御論理／制御回路

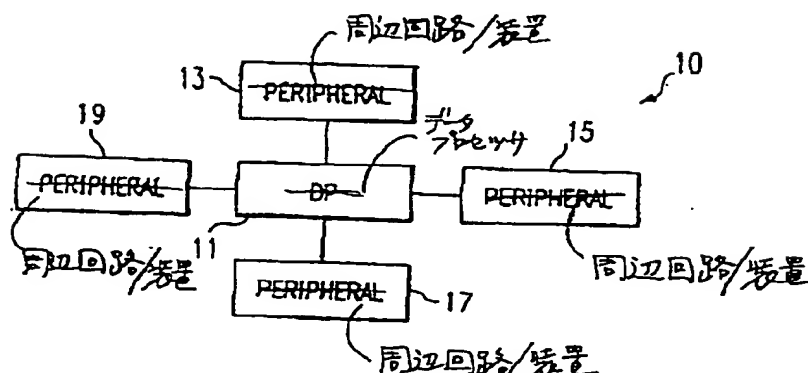
102 制御レジスタファイル

200a, 200b, 211, 212, 213, 214 マルチプレクサ

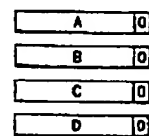
1710, 1720, 1730, 1740 命令取り出しパケット

1725, 1738 分岐命令

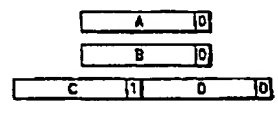
【図 1】



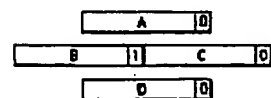
【図 5】



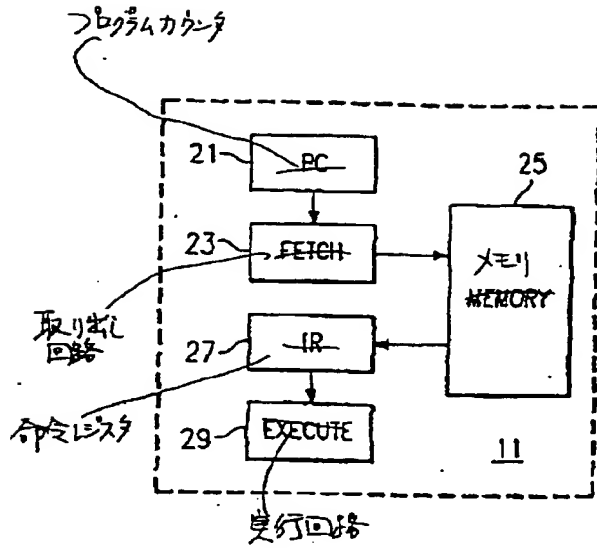
【図 9】



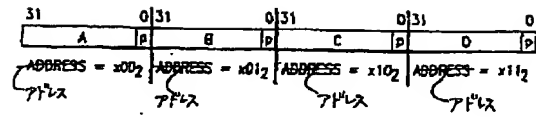
【図 11】



【図 2】



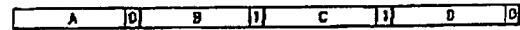
【図 3】



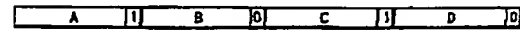
【図 7】



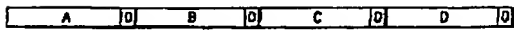
【図 12】



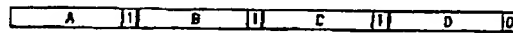
【図 16】



【図 4】



【図 6】



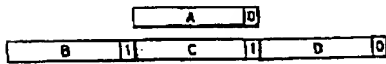
【図 8】



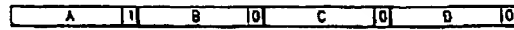
【図 10】



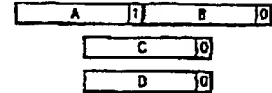
【図 13】



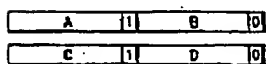
【図 14】



【図 15】



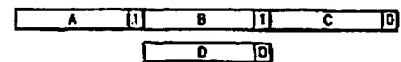
【図 17】



【図 18】

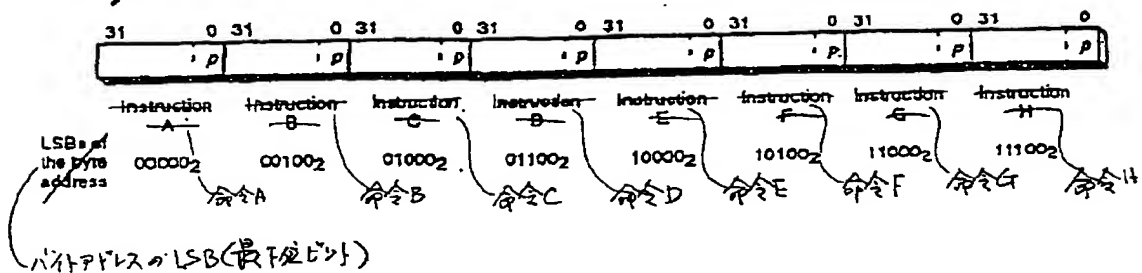


【図 19】





~~Figure 7-5 The Basic Format of a Fetch Packet~~



【図22】

パイフライン フェーズ	Symbol	During this Entire Phase	By the End of this Phase
プログラム アドレス 発生	PG	このフェーズ中に	<p>By the End of this Phase このフェーズの終了時点で</p> <ul style="list-style-type: none"> <li>□ プログラムストアがE1にある場合を除き、PFCおよびPADDDRは、それらの次の値、すなわち、現在のPFCの後の8語境界か、分岐命令の目標アドレスのいずれかにラッチされる。</li> <li>□ プログラムストアがE1にある場合、PFCは変更されずその値のままであり、PADDDRはプログラムストア命令からの値をとる。</li> <li>□ PRSがアクティブの場合、PMSがPADDDRをラッチする。</li> </ul>
プログラム アドレス送信	PS	<ul style="list-style-type: none"> <li>□ PADDDRがPFCが有効である。</li> <li>□ 取り出しパケットを取り出す必要がある場合、PRSがアクティブにセットされる。</li> <li>□ 最終PRSで取り出しパケットが指示された場合、PDSが立ち上げられる。</li> </ul>	<ul style="list-style-type: none"> <li>□ PDSが立ち上げられると、PMSは、PR中に駆動されるように、最後に要求された取り出しパケットをラッチする。</li> </ul>
プログラム 待ち合わせ	PW	<ul style="list-style-type: none"> <li>□ 関連するPW中にPDSがアクティブの場合、PMSは、要求された取り出しパケットをPDATA_1上に駆動する。</li> <li>□ それ以外の場合、PDATA_1上に駆動された前の取り出しパケットが維持される。</li> </ul>	<ul style="list-style-type: none"> <li>□ PW中のアクティブなPDSによつて要求された取り出しパケットが必要なることを決定することができると、CPUはこの取り出しパケットをPDATA_1上にラッチする。それ以外の場合、このラッチは状態を維持する。</li> </ul>
Program Oplo Receive プログラム 受け取り受信	PR		<ul style="list-style-type: none"> <li>□ 取り出しパケットの並列性が検出される。</li> <li>□ 取り出しパケットで次に要求された実行パケットの中の命令が、反対側の機能ユニットにデイスバッチされる。</li> <li>□ 検出されたCRFGフィールドの中でソフトウェア中断コードが検出される。</li> <li>□ 命令：複数サイクルNOP, NOP, SWI, IDLEが、個別の機能ユニットに転送されないため復号化される。</li> <li>□ そのサイクルの終了時点で、DP中の実行パケットのPCCが、PCDCとしてラッチされる。</li> </ul>
Dispatcher ディスパッチ 復号化	DP	PCDCが有効である	
Decode	DC	□ PCDC-valid	□ 実行中のアクションの制御信号が復号化される。

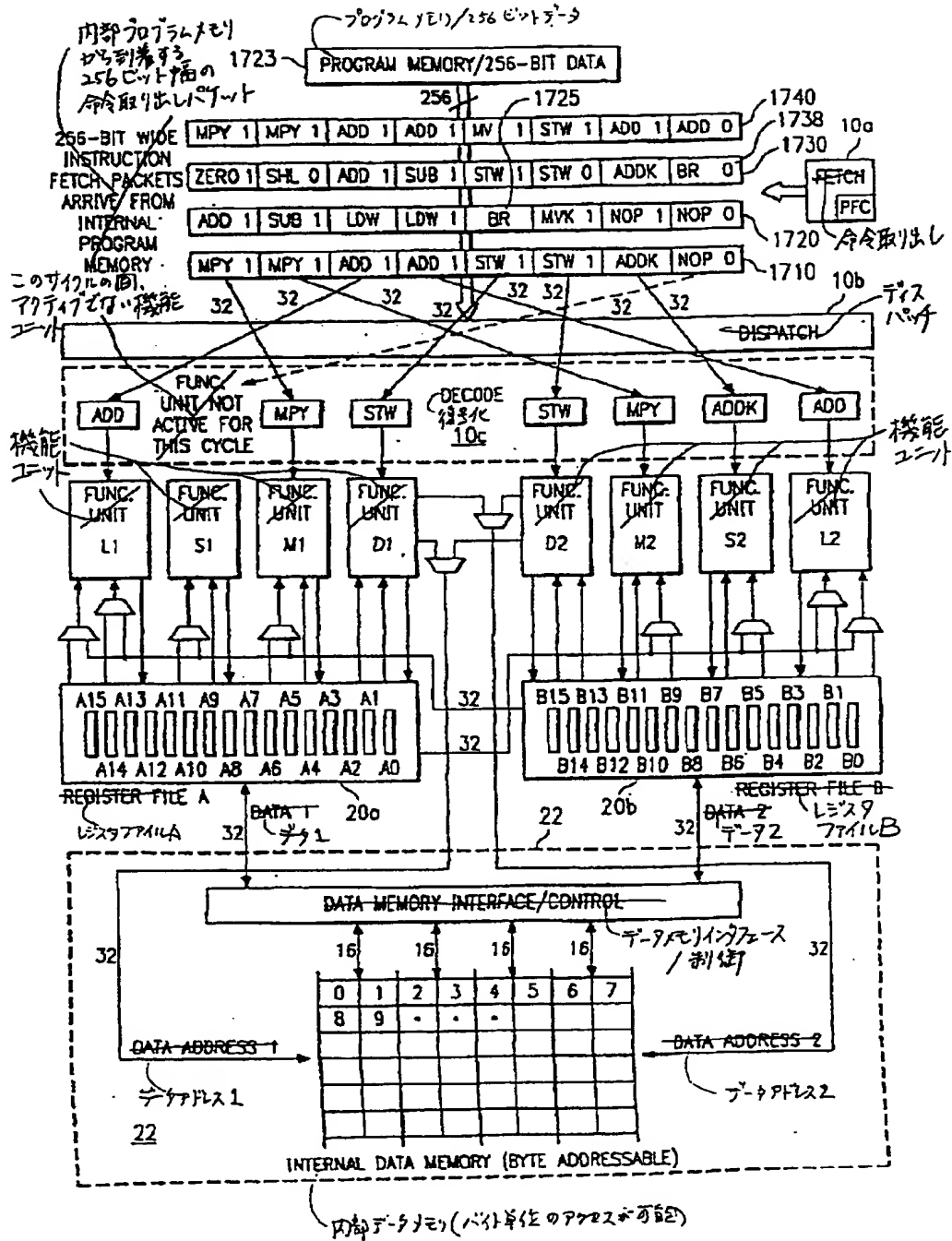
【図23】

Pipeline Phase	Symbol 符号	During this Phase- このフェーズ中 このフェーズで	By the End of this Phase- このフェーズの終了時点で	CPU has Completed Types
Execute 1 実行フェーズ	E1	□ All types: Registers read. 全形式: レジスタが読み出される	<p>□ ISS 形式: 結果がレジスタファイルに書き込まれる。</p> <p>□ ISS 形式: 状態が制御レジスタファイルに書き込まれる。</p> <p>□ LD および ST 形式: アドレス修飾がレジスタファイルに書き込まれる。</p> <p>□ LD および ST 形式: 次のフェーズのため DADDR、DBS および DRNW がラッチされる。</p> <p>□ ST 形式: 次のフェーズのため DDATA_0 がラッチされる。</p> <p>□ BRANCH 形式: PC および PADDR が目標アドレスエラッチする。</p> <p>□ STP 形式: 取り出し中でなければ、PADDR が宛先アドレスをラッチする。次のフェーズのため PWS がアクトにラッチされる。</p> <p>□ LD, ST, STP, IMPY 形式: 中間状態がラッチされる。</p>	<p>□ ISC.</p> <p>□ BR.</p> <p>CPU が終了する形式</p>
Execute 2 実行フェーズ	E2	<p>□ LD および ST 形式: DBS がアクトに PADDR をラッチする。</p> <p>□ LD 形式: DRNW がアクトに PADDR をラッチする。</p> <p>□ ST 形式: DRNW がアクトに PADDR をラッチする。</p> <p>□ DDATA_0 がアクトに PADDR をラッチする。</p>	<p>□ IMPY 形式: 結果がレジスタファイルに書き込まれる。</p> <p>□ IMPY 形式: 状態が制御レジスタファイルに書き込まれる。</p> <p>□ LD および ST 形式: DMS が DADDR をラッチする。</p> <p>□ ST 形式: DMS が DDATA_0 をラッチする。</p> <p>□ STP 形式: DMS が PDATA_0 をラッチする。</p> <p>□ LD 形式: 中間状態がラッチされる。</p>	<p>□ IMPY.</p> <p>□ STD.</p> <p>□ STP.</p>
Execute 3 実行フェーズ	E3		<p>□ LD 形式: DMS は最終ワル中に選択されたデータエラッチする。</p> <p>□ LD 形式: 中間状態がラッチされる。</p>	
Execute 4 実行フェーズ	E4	□ LD 形式: DMS は PDATA_1 上で国連する E2 中に選択されたデータエラッチする。	<p>□ LD 形式: DDATA_1 がラッチされる。</p> <p>□ LD 形式: 中間状態がラッチされる。</p>	
Execute 5 実行フェーズ	E5		□ LD 形式: PDATA_1 がアクトに PADDR をラッチする。でアクトに PADDR、レジスタファイルに書き込まれる。	□ LD.

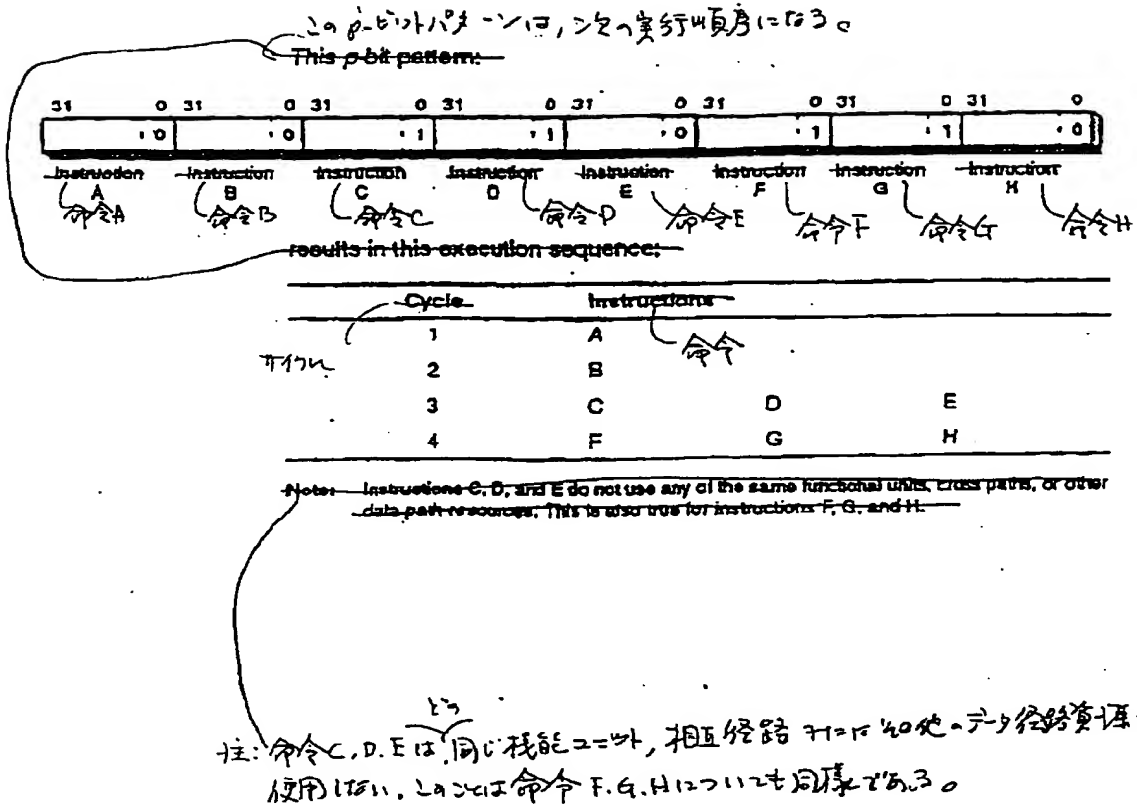




【図25】



【図27】



フロントページの続き

(72)発明者 レイド イー. タトゲ  
アメリカ合衆国 テキサス州リッチモン  
ド, コッパーウッド レーン 1707

(72)発明者 アラン エル. デービス  
アメリカ合衆国 テキサス州シュガーラン  
ド, マラナサ ドライブ 3739

## 【外国語明細書】

**Data Processing With Control Arrangement for Specifying  
Parallel and Sequential Execution of Program Instructions**

5

Cross Reference to Related Applications

10 This application is related to coassigned application serial number  
60/036,482 (TI docket number T-25311) filed contemporaneously herewith and  
incorporated herein by reference.

15

Field of the Invention

This invention relates generally data processing and, more  
particularly, to data processing with both parallel and sequential execution  
of program instructions.

20

Background of the Invention

25

Data processing systems and data processors are used in myriad  
applications which in turn have an impact on virtually every aspect of life.  
The utility of these myriad applications can ordinarily be enhanced by  
increasing the speed and throughput of the associated data processing  
systems and data processors.

30

One way to enhance speed and throughput is, where possible, to  
execute program instructions in parallel rather than in sequential fashion.  
One known approach in this regard is to utilize a special mode instruction

which specifies parallel or sequential execution of program instruction. Another known approach in this regard is to use a mask to specify null instructions in a parallel type packet. Another known solution is to perform data processing in a parallel mode only.

5           Although the aforementioned techniques are capable of improving speed and throughput, they are nevertheless undesirably difficult to implement and disadvantageously costly in terms of processing overhead.

          It is therefore desirable to provide for parallel execution of program instructions in a manner which reduces the implementational difficulties  
10           and processing overhead associated with the above-described approaches.

          An object of the present invention is to utilize a portion of a given program instruction to determine whether that instruction can be executed simultaneously with another program instruction.

### Summary of the Invention

In general, and in a form of the present invention, a data processing device is provided which has circuitry for producing a set of instructions which include instruction portions for indicating whether the respective instructions can be executed simultaneously with another of the instructions. The data processing circuitry includes program execution circuitry connected to the producing circuitry for receiving the set of instructions and selectively executing simultaneously a plurality of the instructions in response to indicator portions.

In another form of the present invention, a method for operating a central processing unit (CPU) within a data processing device comprises the steps of: providing a set of instructions with respective instruction portions for indicating whether the respective instructions can be executed simultaneously; and determining from the instruction portions whether a plurality of the instructions can be executed simultaneously.

Other embodiments of the present invention will be evident from the description and drawings.

### Brief Description of the Drawings

Fig. 1 is a block diagram of a data processing system according to the present invention;

5        Fig. 2 is a block diagram of a portion of the data processing circuitry of Fig. 1;

Fig. 3 illustrates the basic format of an instruction packet utilized in the present invention;

10       Fig. 4 is one example of an instruction packet according to the format of Fig. 3;

Fig. 5 illustrates the execution sequence defined by the instruction packet of Fig. 4;

Fig. 6 illustrates another example of an instruction packet according to the format of Fig. 3;

15       Fig. 7 illustrates the execution sequence defined by the instruction packet of Fig. 6;

Fig. 8 illustrates another example of an instruction packet according to the format of Fig. 3;

20       Fig. 9 illustrates the execution sequence defined by the instruction packet of Fig. 8;

Fig. 10 illustrates another example of an instruction packet according to the format of Fig. 3;

Fig. 11 illustrates the execution sequence defined by the instruction packet of Fig. 10;

25       Fig. 12 illustrates another example of an instruction packet according to the format of Fig. 3;

Fig. 13 illustrates the execution sequence defined by the instruction packet of Fig. 12;

30       Fig. 14 illustrates another example of an instruction packet according to the format of Fig. 3;

Fig. 15 illustrates the execution sequence defined by the instruction packet of Fig. 14;

Fig. 16 illustrates another example of an instruction packet according to the format of Fig. 3;

Fig. 17 illustrates the execution sequence defined by the instruction packet of Fig. 16;

5        Fig. 18 illustrates another example an instruction packet according to the format of Fig. 3;

Fig. 19 illustrates the execution sequence defined by the instruction packet of Fig. 18;

10        Figure 20 is a block diagram of a microprocessor which has an embodiment of the present invention;

Figure 21 is a block diagram of the execution units and register files of the microprocessor of Fig. 20;

Figure 22A is a chart which illustrates the processing phases of an instruction execution pipeline in the microprocessor of Fig. 20;

15        Figure 22B is a chart which illustrates the execution phases of the instruction execution pipeline in the microprocessor of Fig 20;

Figure 23 is a timing diagram which illustrates timing details of processing an instruction fetch packet during the processing phases of Fig. 22A and execution of the execution packet during the execution phases of Fig. 22B;

20        Figure 24 is a block diagram showing instruction dispatching in the microprocessor of Fig. 20;

Figure 25 illustrates the basic format of an instruction fetch packet for the microprocessor of Fig. 20; and

25        Figure 26 illustrates an 8-word fetch packet that is partially parallel.

Corresponding numerals and symbols in the different figures and tables refer to corresponding parts unless otherwise indicated.



### Detailed Description

Fig. 1 is a block diagram of a data processing system 10 according to the present invention. The data processing system 10 includes data processing circuitry 11 and peripheral circuitries 13, 15, 17 and 19. In the exemplary embodiment of Fig. 1, the data processing circuitry 11 is connected to each of the peripheral circuitries 13, 15, 17 and 19 for transfer of information between data processing circuitry 11 and peripheral circuitries 13, 15, 17 and 19. However, and as will be apparent from the following description, a data processing system according to the present invention could include any quantity and type of peripheral circuitries and peripheral devices (such as peripherals 13, 15, 17 and 19) interconnected among themselves and with data processing circuitry 11 in any manner heretofore or hereafter conceivable to workers in the art.

Fig. 2 illustrates a portion of one exemplary embodiment of the data processing circuitry 11 of Fig. 1. In Fig. 2, fetch circuitry 23 accesses memory 25 at an address specified by program counter 21 and causes an instruction packet at that address to be loaded into instruction register 27. Program execution circuitry 29 decodes and executes the instructions of the instruction packet held in instruction register 27.

Fig. 3 illustrates the basic format of the instruction packet fetched from memory 25. In the disclosed exemplary embodiment, an instruction packet includes four 32-bit instructions A, B, C and D. As shown in Fig. 3, instructions A, B, C and D are stored at consecutive addresses in memory 25. Thus, during normal sequential execution of program instructions, instruction A would be executed first, followed sequentially by instructions B, C and D.

Bit 0 of each instruction in Fig. 3 has been designated as a p-bit. The p-bits define how the instructions will be executed. The p-bits of the Fig. 3 instruction packet are inspected from left to right by the program execution circuitry 29. If the p-bit of a given instruction is equal to logic one, then the next sequential instruction in the packet is to be executed in parallel with

the first-mentioned instruction. Program execution circuitry 29 applies this rule until an instruction in the instruction packet is reached with a p-bit equal to logic 0.

5 If a given instruction has a p-bit of 0, then the next sequential instruction is executed sequentially after the given instruction (and after any instructions which are executed in parallel with the given instruction). The program execution circuitry 29 applies this rule until it reaches an instruction in the instruction packet with a p-bit of logic 1.

10 Figs. 4-19 provide application examples of the above-described p-bit rules.

Fig. 4 illustrates an instruction packet in which all p-bits are 0. Thus, instructions A-D are executed sequentially as shown in Fig. 5.

15 Fig. 6 illustrates an instruction packet in which the p-bits of instructions A, B and C are equal to 1, and the p-bit of instruction D is 0. Thus, instructions A, B, C and D are executed simultaneously, that is, in parallel as shown in Fig. 7.

20 In the instruction of Fig. 8, only the p-bit of instruction C is set to one, resulting in the execution sequence of Fig. 9, namely, instructions A and B are executed sequentially, followed by instructions C and D which are executed in parallel.

25 In the instruction packet of Fig. 10, only the p-bit of instruction B is set to one, resulting in the execution sequence shown in Fig. 11, namely instruction A is executed and then followed sequentially by the parallel execution of instructions B and C, which is then followed sequentially by execution of instruction D.

30 In the instruction packet of Fig. 12, the p-bits of instructions B and C are set to one, and the p-bits of instructions A and D are zero. This results in the instruction sequence shown in Fig. 13, namely instruction A is executed and is then sequentially followed by the parallel execution of instructions B, C and D.

In the instruction packet of Fig. 14, only the p-bit of instruction A is set to logic one, resulting in the execution sequence shown in Fig. 15,

namely instructions A and B are executed in parallel and then followed sequentially by the execution of instruction C and then the execution of instruction D.

5 In the instruction packet of Fig. 16, the p-bits of instructions A and C are set to one and the p-bits of instructions B and D are 0, resulting in the execution sequence illustrated in Fig. 17, namely the parallel execution of instructions A and B followed sequentially by the parallel execution of instructions C and D.

10 In the instruction packet of Fig. 18, the p-bits of instructions A and B are set to 1 and the p-bits of instructions C and D are 0. This results in the execution sequence illustrated in Fig. 19, namely instructions A, B and C are executed in parallel and then followed sequentially by execution of instruction D.

15 Because the instruction packet in the disclosed example includes 4 program instructions, the program compiler can always provide instruction D (the fourth instruction) with a p-bit of 0. The compiler determines the values of the remaining p-bits of instructions A, B and C based on the propriety of executing instructions A and B in parallel, the propriety of executing instructions B and C in parallel, and the propriety of executing instructions A, B and C in parallel. For example, if execution of instruction B requires a result provided by execution of instruction A, then the compiler would provide instruction A with a p-bit of 0 so that instruction B would be executed sequentially after instruction A. As another example, if instructions B and C access the same register, then the compiler would provide instruction B with a p-bit of 0 to ensure that instructions B and C are executed sequentially rather than in parallel.

25 Figure 20 is a block diagram of a microprocessor 1 which has an embodiment of the present invention. Microprocessor 1 is a VLIW digital signal processor ("DSP"). In the interest of clarity, Figure 20 only shows those portions of microprocessor 1 that are relevant to an understanding of an embodiment of the present invention. Details of general construction for DSPs are well known, and may be found readily elsewhere. For example,

30

U.S. Patent 5,072,418 issued to Frederick Boutaud, et al, describes a DSP in detail and is incorporated herein by reference. U.S. Patent 5,329,471 issued to Gary Swoboda, et al, describes in detail how to test and emulate a DSP and is incorporated herein by reference. Details of portions of  
5 microprocessor 1 relevant to an embodiment of the present invention are explained in sufficient detail hereinbelow, so as to enable one of ordinary skill in the microprocessor art to make and use the invention.

In microprocessor 1 there are shown a central processing unit (CPU) 10, data memory 22, program memory 23, peripherals 60 and an external  
10 memory interface (EMIF) with a direct memory access (DMA) 61. CPU 10 further has an instruction fetch/decode unit 10a-c, a plurality of execution units, including an arithmetic and load/store unit D1, a multiplier M1, an ALU/shifter unit S1, an arithmetic logic unit ("ALU") L1, a shared multiport register file 20a from which data are read and to which data are  
15 written. Decoded instructions are provided from the instruction fetch/decode unit 10a-c to the functional units D1, M1, S1, and L1 over various sets of control lines which are not shown. Data are provided to/from the register file 20a from/to to load/store units D1 over a first set of busses 32a, to multiplier M1 over a second set of busses 34a, to ALU/shifter unit  
20 S1 over a third set of busses 36a and to ALU L1 over a fourth set of busses 38a. Data are provided to/from the memory 22 from/to the load/store units D1 via a fifth set of busses 40a. Note that the entire data path described above is duplicated with register file 20b and execution units D2, M2, S2, and L2. Instructions are fetched by fetch unit 10a from instruction memory  
25 23 over a set of busses 41. Emulation unit 50 provides access to the internal operation of integrated circuit 1 which can be controlled by an external test system 51.

Note that the memory 22 and memory 23 are shown in Figure 20 to be a part of a microprocessor 1 integrated circuit, the extent of which is  
30 represented by the box 42. The memories 22-23 could just as well be external to the microprocessor 1 integrated circuit 42, or part of it could reside on the integrated circuit 42 and part of it be external to the

integrated circuit 42. Also, an alternate number of execution units can be used.

When microprocessor 1 is incorporated in a data processing system, additional memory or peripherals may be connected to microprocessor 1, as illustrated in Figure 1. For example, Random Access Memory (RAM) 70, a Read Only Memory (ROM) 71 and a Disk 72 are shown connected via an external bus 73. Bus 73 is connected to the External Memory Interface (EMIF) which is part of functional block 61 within microprocessor 42. A Direct Memory Access (DMA) controller is also included within block 61. The DMA controller is generally used to move data between memory and peripherals within microprocessor 1 and memory and peripherals which are external to microprocessor 1.

Several example systems which can benefit from aspects of the present invention are described in U.S. Patent 5,072,418, which was incorporated by reference herein, particularly with reference to Figures 2-18 of U.S. Patent 5,072,418. A microprocessor incorporating an aspect of the present invention to improve performance or reduce cost can be used to further improve the systems described in U.S. Patent 5,072,418. Such systems include, but are not limited to, industrial process controls, automotive vehicle systems, motor controls, robotic control systems, satellite telecommunication systems, echo canceling systems, modems, video imaging systems, speech recognition systems, vocoder-modem systems with encryption, and such.

A description of various architectural features of the microprocessor of Fig. 20 is provided in coassigned application serial number 60/036,487 (TI docket number T-25311). A description of a complete set of instructions for the microprocessor of Fig. 20 is also provided in coassigned application serial number 60/036,487 (TI docket number T-25311).

Figure 21 is a block diagram of the execution units and register files of the microprocessor of Fig. 20 and shows a more detailed view of the buses connecting the various functional blocks. In this figure, all data busses are 32 bits wide, unless otherwise noted. Bus 40a has an address bus DA1

which is driven by mux 200a. This allows an address generated by either load/store unit D1 or D2 to provide an address for loads or stores for register file 20a. Data Bus LD1 loads data from an address in memory 22 specified by address bus DA1 to a register in load unit D1. Unit D1 may manipulate  
 5 the data provided prior to storing it in register file 20a. Likewise, data bus ST1 stores data from register file 20a to memory 22. Load/store unit D1 performs the following operations: 32-bit add, subtract, linear and circular address calculations. Load/store unit D2 operates similarly to unit D1, with the assistance of mux 200b for selecting an address.

10 ALU unit L1 performs the following types of operations: 32/40 bit arithmetic and compare operations; left most 1, 0, bit counting for 32 bits; normalization count for 32 and 40 bits; and logical operations. ALU L1 has input src1 for a 32 bit source operand and input src2 for a second 32 bit source operand. Input msb\_src is an 8 bit value used to form 40 bit source  
 15 operands. ALU L1 has an output dst for a 32 bit destination operands. Output msb\_dst is an 8 bit value used to form 40 bit destination operands. Two 32 bit registers in register file 20a are concatenated to hold a 40 bit operand. Mux 211 is connected to input src1 and allows a 32 bit operand to be obtained from register file 20a via bus 38a or from register file 20b via  
 20 bus 210. Mux 212 is connected to input src2 and allows a 32 bit operand to be obtained from register file 20a via bus 38a or from register file 20b via bus 210. ALU unit L2 operates similarly to unit L1.

ALU/shifter unit S1 performs the following types of operations: 32 bit arithmetic operations; 32/40 bit shifts and 32 bit bit-field operations; 32 bit  
 25 logical operations; branching; and constant generation. ALU S1 has input src1 for a 32 bit source operand and input src2 for a second 32 bit source operand. Input msb\_src is an 8 bit value used to form 40 bit source operands. ALU S1 has an output dst for a 32 bit destination operands. Output msb\_dst is an 8 bit value used to form 40 bit destination operands.  
 30 Mux 213 is connected to input src2 and allows a 32 bit operand to be obtained from register file 20a via bus 36a or from register file 20b via bus

210. ALU unit S2 operates similarly to unit S1, but can additionally perform register transfers to/from the control register file 102.

Multiplier M1 performs 16 x 16 multiplies. Multiplier M1 has input src1 for a 32 bit source operand and input src2 for a 32 bit source operand. ALU S1 has an output dst for a 32 bit destination operands. Mux 214 is  
5 connected to input src2 and allows a 32 bit operand to be obtained from register file 20a via bus 34a or from register file 20b via bus 210. Multiplier M2 operates similarly to multiplier M1.

Figure 22A is a chart which illustrates the processing phases of an  
10 instruction execution pipeline in the microprocessor of Fig. 1. Each phase corresponds roughly to a clock cycle of a system clock. For example, if microprocessor 1 is being operated at 200 MHz, then each phase is nominally 5 Ns. However, in a phase where data is expected from a memory or peripheral, such as RAM 70, the pipeline will stall if the data is  
15 not ready when expected. When stalled, a given pipeline phase will exist for a number of system clock cycles.

In Figure 22A, the first phase of processing an instruction is to generate the program address in phase PG. This is done by loading a program fetch counter PFC which is located in control register file 102.  
20 During the second instruction processing phase PS, an address of an instruction fetch packet is sent to program memory 23 via a program address bus PADDR which is part of bus 41. The third phase PW is a wait phase to allow for access time in memory 23. During the fourth phase PR, a program fetch packet is available from program memory 23 via data bus PDATA\_I which is part of bus 41. During the fifth processing phase DP,  
25 instruction parallelism is detected and instructions that can be executed are dispatched to the appropriate functional units. This aspect of pipeline operation will be described in more detail in later paragraphs. During the sixth processing phase DC, executable instructions are decoded and control  
30 signals are generated to control the various data paths and functional units.

Figure 22B is a chart which illustrates the execution phases of the instruction execution pipeline in the microprocessor of Fig 1. During the first execution phase E1, single cycle instructions, referred to as "ISC", and branch instructions, referred to as "BR", are completed. A designated execution unit performs the operations indicated in Fig. 22B as directed by control circuitry 100. During the second execution phase E2, the following types of instructions are completed by designated execution units under control of control circuitry 100: integer multiply (IMPY), program store instructions (STP), and data store instructions (STD). During the third execution phase E3, execution of load data instructions (LD) continues by latching data from the data memory system (DMS), as indicated. During execution phase E4, the data latched in E3 is transferred to a data input register DDATA\_I in execution unit D1 or D2. During execution phase E5, the LD instruction is completed by manipulating the data in register DDATA\_I and writing the manipulated data to a specified register in register file 20a or 20b.

Figure 23 is a timing diagram which illustrates timing details of processing an instruction fetch packet during the processing phases of Fig. 22A and execution of the execution packet during the execution phases of Fig. 22B. Note that a pipe stall is illustrated in phase PW due to a program memory ready signal PRDY being low in phase PS, and a second pipe stall in phase E3 due a data memory ready signal DRDY being low in phase E2.

Figure 24 is a block diagram showing instruction dispatching in the microprocessor of Fig. 20. In this embodiment, an instruction fetch packet contains eight instructions. Instruction fetch packet 1710 is dispatched and decoded to eight execution units as illustrated. Fetch packet 1720 contains a branch instruction 1725. Instruction fetch packet 1730 contains three instruction execute packets. The first execute packet contains two instructions, ZERO-SHL, which will begin processing in the first delay slot of branch instruction 1725. The second execute packet contains four instruction, ADD-SUB-STW-STW, which will begin processing in the second delay slot of branch instruction 1725. The third execute packet



contains two instructions, ADDK-BR, which will begin processing in the third delay slot of branch instruction 1725.

### Parallel Operations

5        Instructions are always fetched eight at a time. This constitutes a fetch packet. The basic format of a fetch packet is shown in Figure 25. The execution grouping of the fetch packet is specified by the *p*-bit, bit zero, of each instruction. Fetch packets are 8-word aligned.

10        The *p* bit controls the parallel execution of instructions. The *p*-bits are scanned from left to right (lower to higher address). If the *p* bit of instruction *i* is 1, then instruction *i* + 1 is to be executed in parallel with (in the same cycle as) instruction *i*. If the *p*-bit of instruction *i* is 0, then instruction *i* + 1 is executed in the cycle after instruction *i*. All instructions executing in parallel constitute an execute packet. An execute packet can  
15        contain up to eight instructions. All instructions in an execute packet must use a unique functional unit.

      An execute packet cannot cross an 8-word boundary. Therefore, the last *p*-bit in a fetch packet is always set to 0, and each fetch packet starts a new execute packet. As discussed with regard to Figures 4-19, there are  
20        three types of *p*-bit patterns for fetch packets. These three *p*-bit patterns result in the following execution sequences for the eight instructions: Fully serial, Fully parallel, Partially serial.

### Example Parallel Code

25        The || characters signify that an instruction is to execute in parallel with the previous instruction. In the fetch packet of Figure 26, the code would be represented as this:

          instruction A  
30        instruction B  
          instruction C  
      || instruction D  
      || instruction E

```

      instruction F
||   instruction G
||   instruction H

```

5

### Branching Into the Middle of an Execute Packet

If a branch into the middle of an execution packet occurs, all instructions at lower addresses are ignored. In the example in Figure 26, if a branch to the address containing instruction D occurs, then only D and E will execute. Even though instruction C is in the same execute packet, it is ignored. Instructions A and B are also ignored because they are in earlier execute packets.

10

### Resource Constraints

No two instructions within the same execute packet can use the same resources. Also, no two instructions can write to the same register during the same cycle. The following sections describe each of the resources an instruction can use.

15

### Functional Units

Two instructions using the same functional unit cannot be issued in the same execute packet.

20

The following execute packet is invalid:

```

      ADD S1   A0, A1, A2 ; \ .S1 is used for
||   SHR .S1   A3, 15, A4 ; / both instructions

```

25

The following execute packet is valid:

```

      ADD L1   A0, A1, A2 ; \ Two different functional
||   SHR .S1   A3, 15, A4 ; / units are used

```

### Cross Paths (1X and 2X)

30

One unit (either a .S, .L, or .M) per data path, per execute packet, can read a source operand from its opposite register file via the cross paths (1X and 2X). For example, .S1 can read both operands from the A register file, or one operand from the B register file using the 1X cross path. This is denoted by an X following the unit name.

35

Two instructions using the same X cross path between register files cannot be issued in the same execute packet since there is only one path from A to B and one path from B to A.

The following execute packet is invalid:

5           ADD.L1X   A0,B1,A1   ; \ 1X cross path is used  
||   MPY.M1X   A4,B4,A5   ; / for both instructions

The following execute packet is valid:

10           ADD.L1X   A0,B1,A1   ; \ Instructions use the 1x and  
||   MPY.M2X   A4,B4,132   ; /    2x cross paths

The operand will come from a register file opposite of the destination if the x bit in the instruction field is set.

#### 15   Load and Store Path

Loads and stores can use an address pointer from one register file while loading to or storing from the other register file. Two loads and/or stores using an address pointer from the same register file cannot be issued in the same execute packet.

20           The following execute packet is invalid:

||   LDW.D1    \*A0,A1       ; \ Address registers from the same  
||   LDW.D1    \*A2,B2       ; /   register file

The following execute packet is valid:

25           LDW.D1    \*A0,A1       ; \ Address registers from different  
||   LDW.D2    \*B0,B2       ; /   register files

Two loads and/or stores loading to and/or storing from the same register file cannot be issued in the same execute packet.

30           The following execute packet is invalid:

||   LDW.D1    \*A4,A5       ; \ Loading to and storing from the  
||   STW.D2    A6,\*B4       ; /   same register file

The following execute packet is valid:

35           LDW.D1    \*A4,B5       ; \ Loading to, and storing from

|| STW.D2 A6,\*B4 ; / different register files

### Long Paths

5 Only one long result may be written per cycle on each side of the register file. Because the S and L units share a read register port for long source operands and a write register port for long results, only one may be issued per side in an execute packet.

The following execute packet is invalid:

10 || ADD.L1 A5:A4,A1,A3 :A2 ; \ Two long writes  
|| SHL.S1 A8, A9, A7 :A6 ; / on A register file

The following execute packet is valid:

15 || ADD.L1 A5:A4,A1,A3 :A2 ; \ One long write for  
|| SHL.S2 B8, B9, B7 :B6 ; / each register file

Because the L and .S units share their long read port with the store port, operations that read a long value cannot be issued on the L and/or .S units in the same execute packet as a store.

The following execute packet is invalid:

20 || ADD.L1 A5:A4,A1,A3 :A2 ; \ Long read operation and a  
|| STW.D1 A8, \*A9 ; / store

The following execute packet is valid:

25 || ADD.L1 A4, A1, A3 :A2 ; \ No long read with  
|| STW.D1 AB, \*A9 ; / with the store

### Register Reads

More than four reads of the same register cannot occur on the same cycle. Conditional registers are not included in this count.

30 The following code sequence is invalid:

MPY M1 A1,A1,A4 ; five reads of register A1  
|| ADD L1 A1,A1,A5  
|| SUB D1 A1,A2,A3

35 Whereas this code sequence is valid:

MPY M1 A1,A1,A4 ; only four reads of A1

```

|| [A1] ADD      L1  A0,A1,A5
      SUB      D1  A1,A2,A3

```

### Register Writes

5 Multiple writes to the same register on the same cycle can occur if instructions with different latencies writing to the same register are issued on different cycles. For example, an MPY issued on cycle i followed by an ADD on cycle i+1 cannot write to the same register since both instructions will write a result on cycle i+1. Therefore, the following code sequence is invalid:

```

      MPY      M1  A0,A1,A2
      ADD      L1  A4,A5,A2

```

### Detectability of Write Conflicts

15 The following sequence of execute packets shows different multiple write conflicts. For example, the ADD and SUB in execute packet L1 write to the same register. This conflict is easily detectable.

```

20 L1:      ADD.L2  B5,B6,B7 ; detectable, conflict
    ||      SUB.S2  B8,B9,B7
    L2:      MPY.M2  B0,B1,B2 ; \ not detectable
    L3:      ADD.L2  B3,B4,B2 ; /
    L4: [B0]  ADD.L2  B5,B6,B7 ; detectable, no conflict
    || [B0]  SUB.S2  B8,B9,B7
25 L5: [B1]  ADD.L2  B5,B6,B7 ; \ not detectable
    || [B0]  SUB.S2  B8,B9,B7 ; /

```

30 The MPY in packet L2 and the ADD in packet L3 might both write to B2 simultaneously; however, if a branch instruction causes the execute packet after L2 to be something other than L3, this would not be a conflict. Thus, the potential conflict in L2 and L3 might not be detected by the assembler. The instructions in L4 do not constitute a write conflict because they are mutually exclusive. In contrast, because it is not obvious that the instructions in L5 are mutually exclusive, the assembler cannot determine

35 a conflict. If the pipeline does receive commands to perform multiple writes to the same register, the result is undefined.

Although exemplary embodiments of the present invention are described above, this does not limit the scope of the invention, which can be practiced in a variety of embodiments.

**WHAT IS CLAIMED IS:**

1. A data processing system, comprising:

5 data processing circuitry having circuitry for producing a set of instructions which include respective instruction portions for indicating whether the respective instructions can be executed simultaneously with another of the instructions; and

10 said data processing circuitry including program execution circuitry connected to said producing circuitry for receiving the set of instructions and selectively responsive to said instruction portions for executing simultaneously a plurality of said instructions indicated by said instruction portions.

15 2. A method of processing program instructions in a data processing system, comprising the steps of:

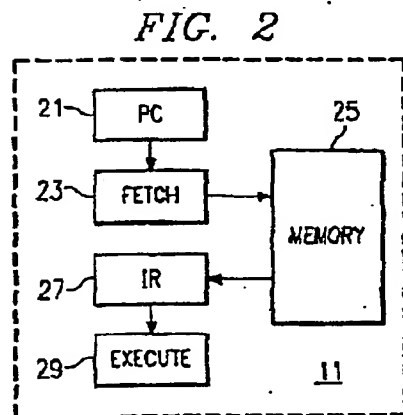
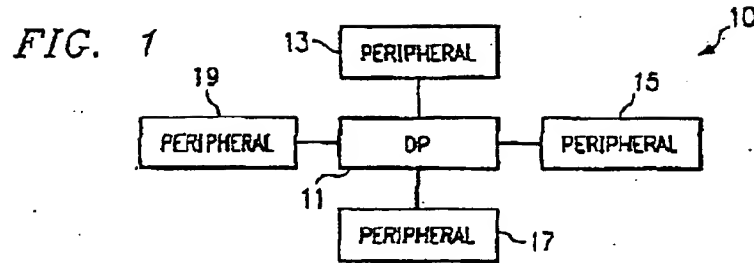
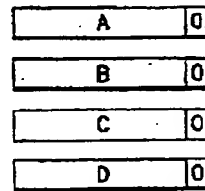
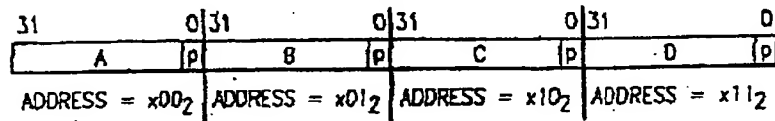
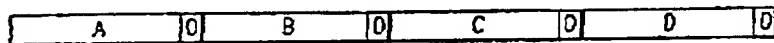
providing a set of instructions with respective instruction portions for indicating whether the respective instructions can be executed simultaneously with another of the instructions; and

20 determining from said instruction portions whether a plurality of said instructions can be executed simultaneously.

3. A method of compiling a program for execution by a data processing system, comprising the steps of:

25 determining whether a first program instruction can be executed simultaneously with a second program instruction that immediately sequentially follows the first program instruction in program; and

30 providing the first instruction with an instruction portion that indicates whether the first instruction can be executed simultaneously with the second instruction.

**FIG. 5****FIG. 3****FIG. 4****FIG. 6****FIG. 7**



2

11-25509  
2 of 910

FIG. 8



FIG. 9

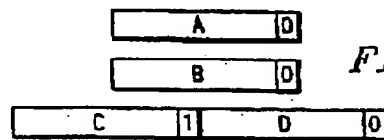


FIG. 10



FIG. 11

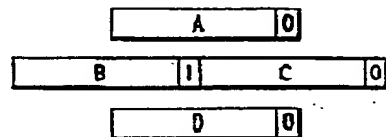
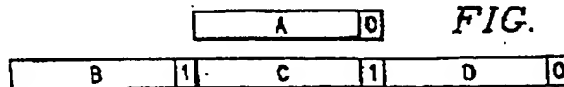


FIG. 12



FIG. 13



3

11-25309  
3 of 810

FIG. 14

A	1	B	0	C	0	D	0
---	---	---	---	---	---	---	---

A	1	B	0
---	---	---	---

C	0
---	---

D	0
---	---

FIG. 15

FIG. 16

A	1	B	0	C	1	D	0
---	---	---	---	---	---	---	---

A	1	B	0
---	---	---	---

C	1	D	0
---	---	---	---

FIG. 17

FIG. 18

A	1	B	1	C	0	D	0
---	---	---	---	---	---	---	---

A	1	B	1	C	0
---	---	---	---	---	---

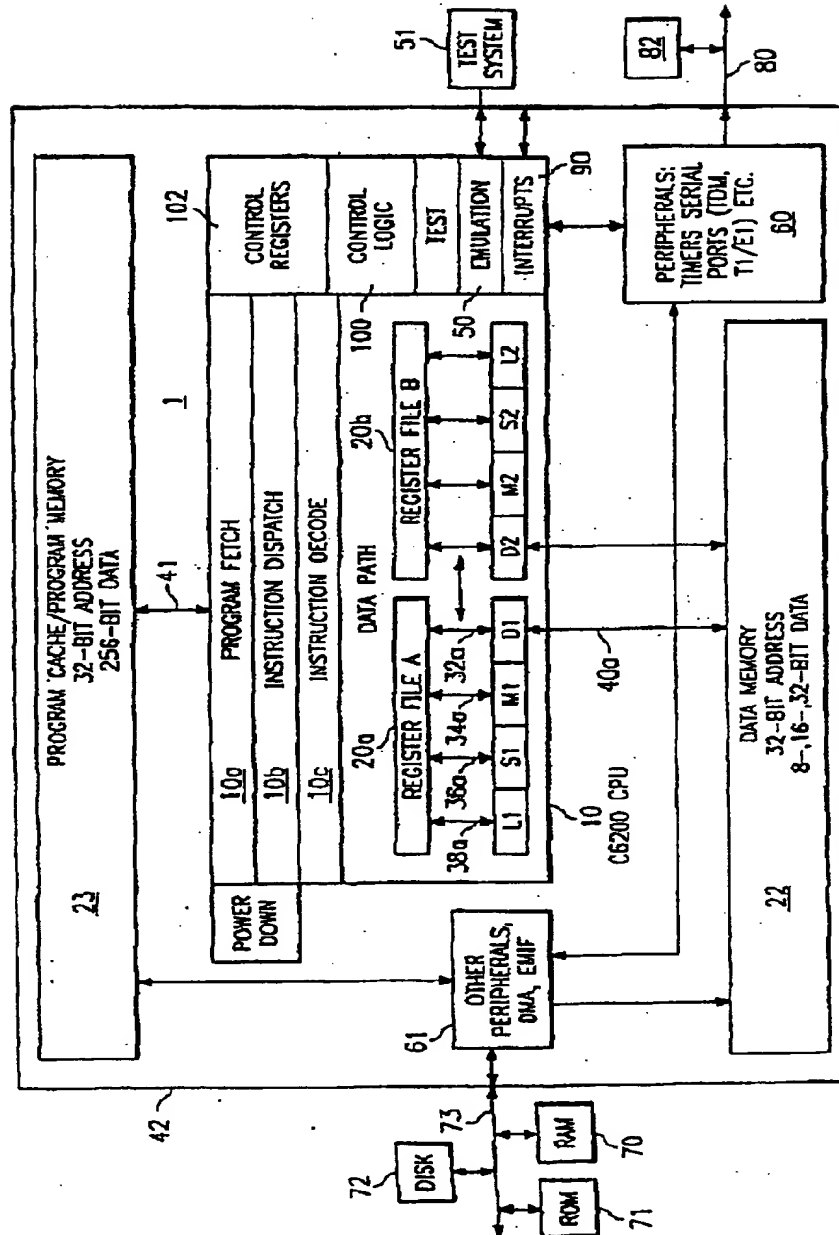
D	0
---	---

FIG. 19

4

11-25309  
4 of 8/2

FIG. 20



5

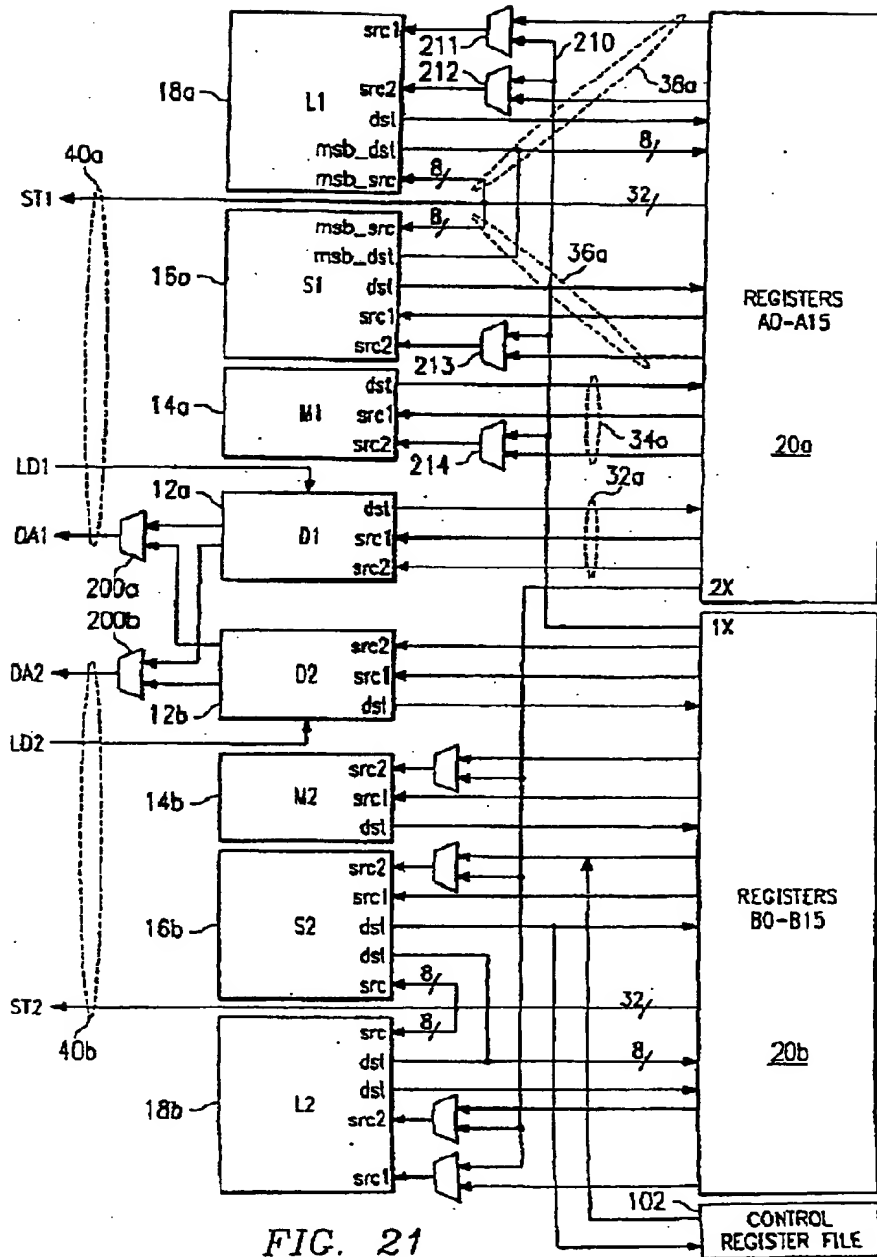
11-25309  
5 of 8/0

FIG. 21

6

11-25309  
6 of 8/0

FIG. 22A

Pipeline Phase	Symbol	During this Entire Phase	By the End of This Phase
Program Address Generate	PG		<input type="checkbox"/> Except when a program store is in E1, PFC and PADDR and latched to their next values; either the next 8-word boundary after the current PFC or a branch target address. <input type="checkbox"/> When a program store is in E1, the PFC remains unchanged and PADDR takes the value from the program store instruction.
Program Address Send	PS	<input type="checkbox"/> PFC valid on PADDR. <input type="checkbox"/> If a fetch packet fetch needed, PRS set active.	<input type="checkbox"/> If PRS active, PMS latches PADDR.
Program Wait	PW	<input type="checkbox"/> If the fetch packet indicated by the last PRS is necessary, PDS is asserted.	<input type="checkbox"/> If PDS is asserted, PMS latches the last requested fetch packet to be driven during PR.
Program Data Receive	PR	<input type="checkbox"/> If PDS active during the associated PW, PMS drives requested fetch packet onto PDATA_1. Otherwise the previous fetch packet driven on to PDATA_1 is maintained.	<input type="checkbox"/> If it can be determined that the fetch packet requested by an active PDS in PW is needed, CPU latches this fetch packet on PDATA_1. Otherwise, this latch maintains state.
Dispatch	DP		<input type="checkbox"/> Parallelism of the fetch packet detected. <input type="checkbox"/> Instructions in the next requested execute packet in the fetch packet dispatched to the appropriate functional units. <input type="checkbox"/> Detection of the software breakpoint code in the CRCS field detected. <input type="checkbox"/> Multi-cycle NOP, MOP, SWI, and IDLE decoded because they are not passed to individual functional units. <input type="checkbox"/> PC of the execute packet in DP latched at the end of the cycle as PCDC.
Decode	DC	<input type="checkbox"/> PCDC valid.	<input type="checkbox"/> Control signals for actions during execute decoded.

11-25505  
7 of 9/0

FIG. 22B

Pipeline Phase	Symbol	During this Phase	By the End of This Phase	CPU has Completed Types
Execute 1	E1	<input type="checkbox"/> All types: Registers read.	<input type="checkbox"/> Type ISS: Results written to the register file. <input type="checkbox"/> Type ISS: Status written to the control register file. <input type="checkbox"/> Type LD and ST: Address modifications written to the register file. <input type="checkbox"/> Type LD and ST: DADDR, DBS, and DRNW latched for next phase. <input type="checkbox"/> Type ST: DDATA_0 latched for next phase. <input type="checkbox"/> Type BRANCH: PFC and PADDR latches the target address. <input type="checkbox"/> Type STP: PADDR latches the destination address if no fetch is pending. PWS latched as active for next phase. <input type="checkbox"/> Types LD, ST, STP, IMPY: Intermediate states latched.	<input type="checkbox"/> ISC. <input type="checkbox"/> BR.
Execute 2	E2	<input type="checkbox"/> Types LD and ST: DBS active and DADDR valid. <input type="checkbox"/> Type LD: DRNW active. <input type="checkbox"/> Type ST: DRNW inactive and DDATA_0 valid.	<input type="checkbox"/> Type IMPY: Results written to the register file. <sup>3</sup> <input type="checkbox"/> Type IMPY: Status written to the control register file. <input type="checkbox"/> Types LD and ST: DMS latches ADDR <input type="checkbox"/> Type ST: DMS latches DDATA_0. <input type="checkbox"/> Type STP: PWS latches PDATA_0. <input type="checkbox"/> Type LD: Intermediate states latched.	<input type="checkbox"/> IMPY. <input type="checkbox"/> STD. <input type="checkbox"/> STP.
Execute 3	E3		<input type="checkbox"/> Type LD: DMS latches data requested during the last cycle. <input type="checkbox"/> Types LD: Intermediate states latched.	
Execute 4	E4	<input type="checkbox"/> Type LD: DMS drives on DDATA_1 data requested during associated E2.	<input type="checkbox"/> Type LD: DDATA_1 latched. <input type="checkbox"/> Types LD: Intermediate states latched.	
Execute 5	E5		<input type="checkbox"/> Type LD: DDATA_1 right-aligned, either sign-extended or zero-filled, and written to the register file.	<input type="checkbox"/> LD.

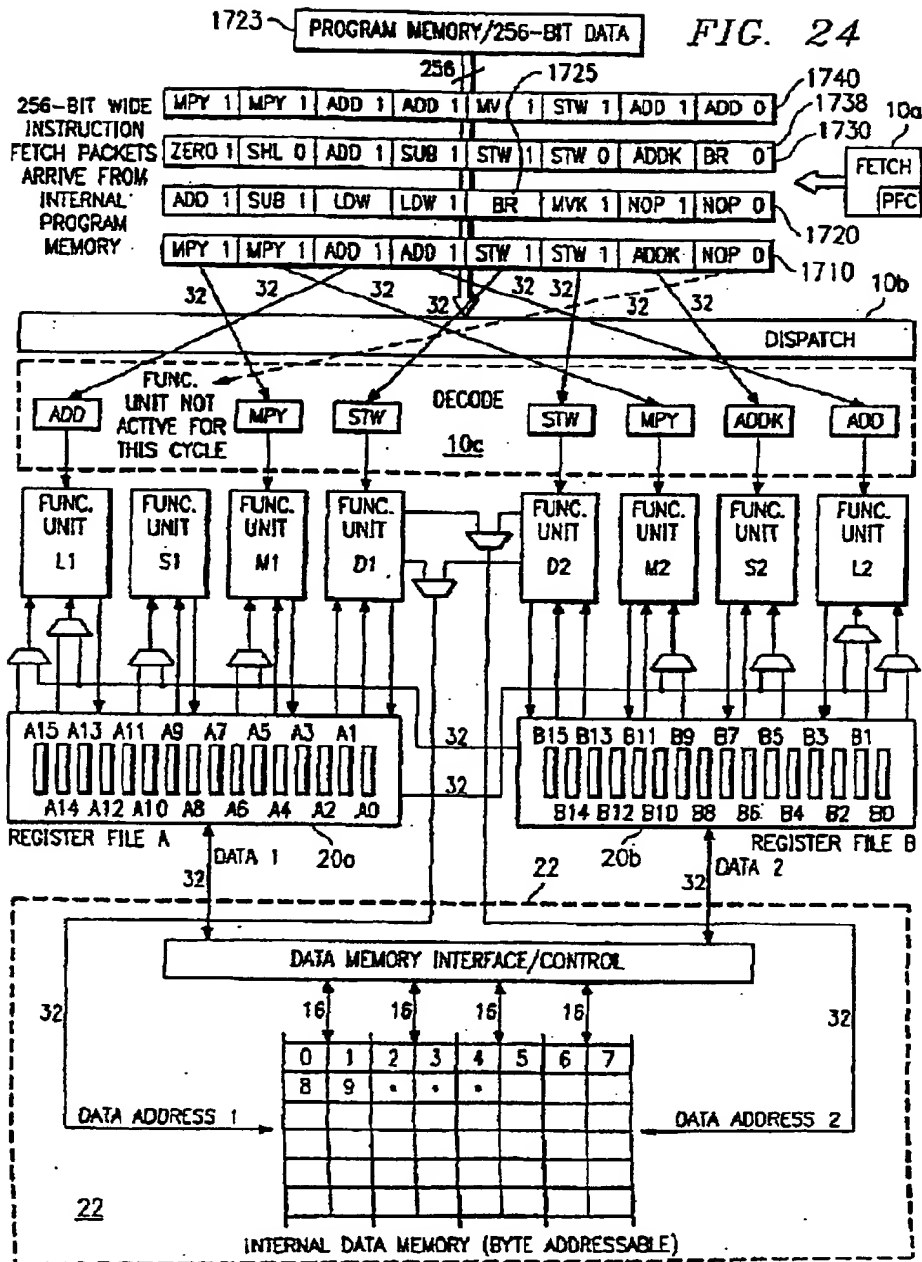
11-25509  
8 of 8

FIG. 23

CLOCK	1	2	3	4	5	6	7	8	9	10	11	12	13
PIPELINE	1	2	3	4	5	6	7	8	9	10	11	12	13
PHASE	PG	PS	PW	PR	DP	DC	ET	E2	E3	E4	E5		
PROY (FROM PMS)													
DRDY (FROM DMS)													
RDY EXTERNAL													
RDY INTERNAL													
PRS													
PDS													
PADDR													
PDATA_1													
PCDC													
DRNW													
DBS													
DADDR													
DDATA_1													
DDATA_0													

T1-25309  
9 of 8/0

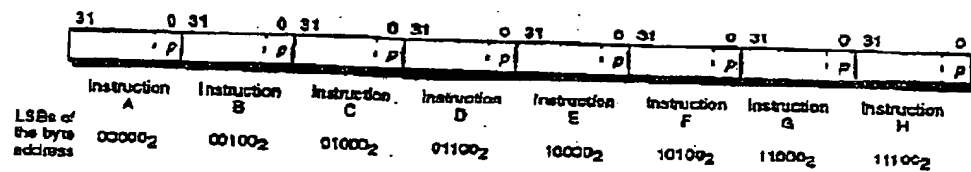
9



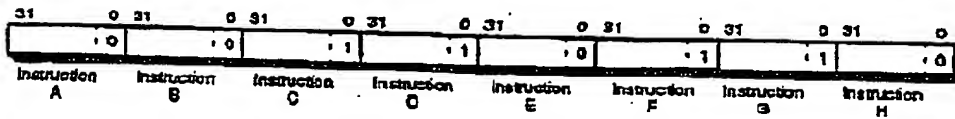


TI-25309<sup>10</sup>  
1/10

Fig. 25

~~Figure 25~~ The Basic Format of a Fetch Packet

This p-bit pattern:



results in this execution sequence:

Cycle	Instructions
1	A
2	B
3	C D E
4	F G H

Note: Instructions C, D, and E do not use any of the same functional units, cross paths, or other data path resources. This is also true for instructions F, G, and H.

Figure 26

## 1. Abstract

A data processing system 10 includes data processing circuitry 11 having circuitry (21, 23 and 25) for producing a set of instructions which include respective instruction portions for indicating whether the  
5        respective instructions can be executed simultaneously with another of the instructions. Program execution circuitry (29) receives the set of instructions and is selectively responsive to the instructive portions for executing simultaneously a plurality of the instructions indicated by the  
10        instruction portions.

## 2 Representative Drawing

Fig 2